The Role of Advanced Penetration Testing Techniques in Enhancing Cybersecurity: A Survey on Web Application Security

Emmanuel Bugingo [1,*)], Voltaire Ishimwe [2)], Ghislaine Uwase Simbi [3)], Adeline Dusenge [4)], and Jean Baptiste Nizeyimana [5)]

[1, 2, 3, 4, 5)] School of Computing and Information Technology, University of Kigali
Kigali, Rwanda

[1)] College of Business and Economics, University of Rwanda
Huye, Rwanda

E-mail: ebugingo@uok.ac.rw[1)], voltaireishimwe@gmail.com[2)], ghislainerwema7@gmail.com[3)], dusengead@gmail.com[4)], and nizeyimanaj215@gmail.com[5)]

ABSTRACT

The Internet continues to grow as an exciting platform for numerous web applications, providing crucial services to various industries and user communities. However, this expansion comes with an increase in cybersecurity threats, as web applications continue to be a primary target for malicious entities. Despite the creation of numerous security frameworks, yearly reports and the OWASP (Open Web Application Security Project) Top 10 consistently highlight the ongoing presence of severe vulnerabilities in contemporary web platforms. This paper examines the significance of advanced penetration testing methods in enhancing cybersecurity, particularly in the context of web application security. Utilizing a combination of manual and automated testing approaches, incorporating tools such as BURP SUITE (a web security testing tool) and METASPLOIT (an exploitation platform), this study examines how sophisticated penetration testing can uncover, exploit, and address vulnerabilities, including Structured Query Language (SQL) injections, cross-site scripting (XSS), and weak authentication methods. The results highlight that even small vulnerabilities can have significant practical impacts, emphasizing the importance of ongoing, intelligent testing approaches.

Keywords: Penetration testing, OWASP, manual testing, automated testing, burp suite, metasploit, secure software development.

1. Introduction

In the era of information and digitalization, web applications are essential. The number of people using the Internet and web applications has increased significantly in recent years. Due to their nature, web applications and services are prone to security risks because they are connected to the Internet [1]. As the Internet and web applications become increasingly widespread, every web application requires an adequate level of security to prevent cyberattacks and store information securely. Various logical and technical vulnerabilities can affect web applications. Furthermore, developing a secure web application without vulnerabilities is a challenge for security analysts. This makes web applications an attractive target for cyber-attacks, with an average of 10 million web attacks per day [2], a figure that will only increase the number of users of web applications increases [3]. A WAV is defined as “a flaw in the application that stems from coding defects and causes severe damage to the application upon exploitation” [4]. To identify and mitigate vulnerabilities that may be exploited by attackers, a penetration testing method or ethical hacking is used [5]. The Open Web Application Security Project (OWASP) provides the standard for such penetration testing methodology to test web applications and could be used to evaluate the effectiveness of web vulnerability scanners [6], [7]. Web Application Vulnerability Scanners (WAVS) are tools used by penetration testers. It is used to conduct web application evaluations with the primary goal of identifying and mitigating potential

* Corresponding author.
Received: August 13th, 2025. Revised: November 30th, 2025. Accepted: December 1st, 2025.
Available online: January 15th, 2026.

DOI: https://doi.org/10.12962/j24068535.v24i1.a1372

vulnerabilities to prevent security breaches. Security measures should be incorporated throughout the development life cycle rather than being added and tested only at the final stages of the development life cycle [8]. The challenge here is that when using different WAVS, they will not offer the same vulnerability results for the same target [8], [9]. These differences arise from the different levels of precision, speed, and coverage in terms of finding various vulnerabilities, for example, with the respect to XSS and SQLi attacks [8], [10]. These differing results mean that penetration testers need to utilize multiple WAVS and thus, the accuracy and detection coverage of the penetration testing report depend on the testers' knowledge and experience of the most effective WAVS in particular situations. Technical vulnerabilities include structured query language (SQL) injection, cross-site scripting (XSS), remote file inclusion, and local file inclusion. These vulnerabilities compromise the security of web applications and are mainly caused by poor programming practices or outdated systems. Web applications go through rapid development phases with short turnaround times, making it a challenge to eliminate vulnerabilities. Despite the difficulty of removing these vulnerabilities, it is essential to ensure that any web applications are kept up-to-date and securely coded. A penetration test, a technique used to gain access to a system and secure it, can be employed to analyze the vulnerability of a web application. To conduct this type of testing legally, we must ask permission from the application's owner; as a result, penetration testing is effective in addressing network security issues. Furthermore, web penetration testing refers to testing web-based applications, including thin-client applications, file transfers, appliances, and portals, to discover vulnerabilities prone to exploitation, verify the presence of appropriate controls, and conduct probing and vulnerability analyses [11]. To stay safe against cyber-attacks, penetration testing can be used to assess the effectiveness and ineffectiveness of web application security arrangements.

In a penetration test, the target systems and the goal are identified, and then the information is reviewed, followed by measures taken to achieve the goals. By performing a penetration test, one can determine if a system is vulnerable to attack if its defenses are adequate, and if any defenses have been overcome. This information is paired with an accurate assessment of potential impacts on the organization and a range of technical and procedural countermeasures to mitigate them [12]. Once the vulnerability scan of a host has been completed, the results of these tests or attacks are documented and presented as a report to the system owner. A penetration test can be conducted using various tools, so selecting the right one is very important.

During penetration testing, it is essential to implement secure coding practices, educate web application developers, and implement automated security scans. Therefore, organizations can be better prepared to prevent malicious attacks [13]. Moreover, during penetration testing, security measures must be enhanced to prevent malicious actors from exploiting these vulnerabilities. In the case of [14], although particular security measures, such as encryption and authentication procedures, were implemented, they were not fully utilized. Additionally, other steps, such as the use of firewalls and intrusion detection systems, could have been taken to strengthen the security of the websites. These security procedures could have provided a stronger defense against potential attacks and alerted the company to any suspicious activity on its websites. Additionally, the use of secure coding practices and regular security audits was recommended to ensure that any newly discovered vulnerabilities would be identified and patched promptly. The main contributions of this paper are summarized as follows:

- Presents a comprehensive review of penetration testing approaches and tools for web application analysis, including an examination of previous literature, the advantages and limitations of each proposed solution, and recommendations for selecting effective tools.
- It discusses current security challenges in web applications, the importance of in-depth testing before deployment, and provides insights and future research directions beneficial to both penetration testers and researchers.

The rest of this paper is organized as follows. In Section 2, establishes the background of web penetration testing. In Section 3, reviews relevant literature on security vulnerabilities and testing approaches. In Section 4, details our research methodology. In Section 5, examination of critical web application vulnerabilities. In section 6 analyses current protection mechanisms and takeaway message. Finally, Section 7 concludes with key findings and recommendations for future research.

Table 1: Manual vs. automated penetration tests.

Criteria	Manual Penetration Testing	Automated Penetration Testing
Testing process	The process is manual, non-standard, and capital-intensive, making it expensive to customize.	Easy to use, provides clear, actionable reports, and eliminates errors and tedious manual tasks.
Network modification	Results in numerous system modifications.	There is no change in the systems.
Exploiting development and management	Maintaining an exploit database is time-consuming and requires considerable expertise. To achieve cross-platform functionality, it is necessary to rewrite and port code.	All exploits are developed and maintained by the product vendor. For maximum effectiveness, exploits are continuously updated. These exploits have been professionally designed and thoroughly tested, and they are safe to use. Various platforms and attack vectors are used in the development of exploits.
Reporting	It requires significant effort to record and collate all results manually. It also requires all reports to be manually generated.	Someone can customize the report and include comprehensive histories and findings.
Clean-up	Every time a vulnerability is discovered, the tester must manually undo the changes.	Automated testing products offer clean-up solutions.
Logging/Auditing	The process is slow, cumbersome, and often inaccurate.	All activities are automatically recorded.
Training	Testing methods that are not standardized and ad hoc must be learned by testers.	Testing with automated tools is easier than testing manually.

2. Background

2.1. Penetration Testing

Penetration testing involves evaluating the security of a network or system through simulated attacks conducted by authorized ethical hackers. Penetration testing differs from malicious hacking in that it is authorized by the organization, typically through a signed agreement, and the results are compiled into a detailed report. Penetration tests are designed to improve data security [1], and the information and weaknesses that they identify remain confidential until all deficiencies have been resolved [15]. As part of penetration testing, developers assess system weaknesses and possible attack vectors to build a secure system aligned with the organization's objectives. It is essential for organizations to track and evaluate the severity of security issues.

A penetration test can be carried out either manually or through automated tools. Manual testing demands that a proficient team of testers manage the entire process, with their physical presence necessary during the assessment. As a result, this method is often considered expensive. An automated penetration test offers a simple and reliable method for carrying out all testing activities. In addition, since most of the work is automated, it is more time-efficient. A further advantage of this type of test is that the parameters can be reused.

Table 1 highlights that manual penetration testing requires highly skilled specialists with extensive security expertise. This approach is both resource-intensive and costly, and recruiting qualified testers can be challenging. By integrating expert knowledge with technology, penetration testers can develop automated tools that streamline testing processes. These automated tools also enable non-specialist users to scan systems and obtain a comprehensive overview of an organization's security posture.

Traditionally, manual penetration testing has been a key component of application security. While it provides thorough and effective assessments, the process is time-intensive and costly, limiting both its practicality and scalability. Automated penetration testing offers a more cost-efficient approach, allowing security evaluations to be incorporated into software development lifecycles. Although professional oversight may still be required to monitor

these tools, automation enables the scanning of many more applications in less time, reducing expenses and helping to meet development deadlines.

2.2. Related work

Security testing and various related issues, such as privacy, are generally articulated in recent survey articles[16], [17], [18], [19], [20], [21] introduced various tools, including Sparta, Nmap (Network Mapper), Zenmap, Netcraft, IP Address Tracking, and VirusTotal, for information gathering. During the practical experimentation of using security tools on a specific application, it was observed that the OWASP ZAP tool was able to detect both medium- to high-level risks and low-level risks. The medium- to high-level severity vulnerabilities identified are URL rewriting, Application error disclosure, the X-Frame-Options header, and SQL injection. Priyanka et al. [22] presented three web application vulnerabilities such as SQL injection, XSS (Cross-site scripting), and CSRF (Cross-site request forgery). It also discusses the tools that can be used for VAPT. The study concluded that effective prevention of SQL injection attacks involves validating and sanitizing user inputs, implementing whitelists, using prepared statements to pass input data as parameters, and ensuring that sensitive information is not exposed through error messages on web pages. Amin et al. [23] Focus on how the Red Team performs security assessments of the applications. There are three types of teams: 1) the red team, 2) the purple team, and 3) the blue team. The red team imitates the activities that an attacker would perform by using tools and techniques to conduct a real-world attack, just as an attacker would. The blue team represents the internal cybersecurity team of an organization, charged with defending systems against simulated attacks by the red team as well as real threats from both inside and outside the organization. The purple team blends blue team defense techniques with red team attack tactics to enhance overall security. The paper also outlined several rules of engagement for red team operations. These include carrying out the assigned activities, adhering to all applicable laws, regulations, and organizational policies, and following the team's established operational methodology. Additional responsibilities involve identifying inputs to the target environment, developing or refining exploit tools for testing purposes, conducting OSINT, and detecting actions that expose system weaknesses. Red team members must also support the team lead in preparing the final engagement report and participate in physical security assessments under the team lead's direction.

Vats et al. [24] starts by discussing various penetration-testing strategies, including external penetration-testing, internal penetration-testing, blind penetration-testing, double-blind penetration-testing, and targeted penetration-testing. Penetration tests fall into three major types black box, gray box, and white box each offering a different level of visibility into the system being evaluated. Goutam et al. [25] proposed a framework designed to enhance the security of applications used by financial institutions. The proposed algorithm initiates its process by prompting the user to supply their login ID and password. When the entered credentials correspond to a registered user, a one-time password is issued to both the associated mobile number and email ID. Once the OTP is confirmed, the system prompts the user to enter the reference number. If the reference number is not supplied, the system redirects the user back to the login page. This reference number is generated automatically during initial registration and is unique to each customer; however, it is not stored in the main user-details database. The proposed framework was implemented and tested on a financial application, where it proved effective in identifying several vulnerabilities. These included clickjacking risks due to missing X-Frame-Options headers, disabled cross-site scripting protections, SQL injection points, and exposure of private IP addresses all of which could be exploited by attackers to launch further attacks on the system. The functionalities of VAPT, testing checklists for assessing applications, OWASP's top 10 security risks, and the workflow of penetration testing are presented [26]. The authors presented a checklist that should be followed when practicing secure coding, which includes input validation, authentication, password management, output encoding, access control, session management, communication security, cryptographic practices, error handling and logging, system configuration, file management, database security, and memory management. Khera et al [27]. performed an analysis of the lifecycle of the VAPT process. It also shortlists some useful VAPT tools for testing and identifying vulnerabilities in the target system. It explains the need for organizations to adopt vulnerability assessments and penetration testing at various levels to prevent cyberattacks. It also discussed vulnerability assessment and penetration testing in detail. The primary reasons for these vulnerabilities to arise are system misconfiguration, weak password combinations, a system connected to an insecure network, and poorly designed

software and hardware. Hasan and Meva[28]. Discussed in detail the VAPT process model, its benefits, and the tools used in the process. The paper focuses on high-risk vulnerabilities like SQL injection, local file inclusion, cross-site scripting, and remote file inclusion.

Hasan and Meva [28] also performed a literature survey to perform an analysis of the generalized VAPT process used among all of them and the tools that are helpful when performing vulnerability assessment and penetration testing. Yaqoob et al. [29] presented vulnerability assessment and why it needs to be performed in every organization. It also discussed common network vulnerabilities, threats to the vulnerable networks, and the vulnerability management lifecycle. Additionally, the paper presents the penetration testing process and compares it with the vulnerability assessment process. Hasan et al. [30], [31] analyzed the different approaches to perform vulnerability assessment and penetration testing in web applications to ensure secure web applications in the ever-evolving cyber world. First, the paper [30] discussed the software development life cycle and its phases, including planning, analysis, design, implementation, testing, integration, and maintenance, in a detailed manner. Next, the paper divides the vulnerability into two categories: logical vulnerability and technical vulnerability. Paper [31] discussed penetration testing in detail, including its step-by-step approach, its benefits for securing the network, and the tools used for performing penetration testing. Penetration testing is an effective method for identifying and assessing system vulnerabilities.

Haque et al. [32] presented ways to secure web services, the challenges faced in securing web services, and recommendations to overcome those security challenges in web services. It also discusses the 10 most common vulnerabilities and presents ways to prevent three of these vulnerabilities, such as SQL injection, cross-site scripting, session management, and broken authentication.

Similarly, common vulnerabilities are discussed in the studies [33], [34]. Security controls are presented. To illustrate this, a livestock data center is used as a case study to perform the assessment and testing and to propose relevant security controls [35]. Singh et al. [36] presented methodology of performing penetration testing. It describes what penetration testing is, its various techniques, and the reasons for performing penetration testing. Goel et al. [26], [37], [38], [39], [40], [41] presented the VAPT lifecycle as a procedure to be performed on the web application infrastructure, which can help prevent cyberattacks. A study [34-40] shows that the exploration of vulnerabilities depends on the type of programming environment and application specifications. All the afore-mentioned studies used automated vulnerability scanning methods and tools. The alternative to automated testing is machine learning testing, which is often the best option for modern applications. Furthermore, these studies primarily focused on discussing or presenting a limited number of web application vulnerabilities.

3. Literature Review

Mirjalili et al. [1] proposed the design and development of a distributed framework to automate web pentation testing, the major components of which are an operational unit called an executor that conducts attacks and a control unit called an orchestrator that orchestrates them across consecutive stages. The authors outlined the core activities involved in penetration testing and introduced a method for coordinating the attackers responsible for carrying out these tasks. They also proposed a flexible approach for integrating external tools by mapping identified vulnerabilities to the framework's available resources. To fully demonstrate the capabilities of this distributed system, the authors developed a collection of tools with a web-based interface that integrates smoothly into the framework. The proposed framework offers notable benefits, including scalability, distributed operation, and user-friendly design, all of which enable rapid creation and deployment of attack scenarios. It provides users with the ability to access, modify, and incorporate tools with minimal effort, thereby enhancing the effectiveness of their attacks and allowing quick adaptation to changing conditions or newly discovered vulnerabilities. This adaptability gives users greater control and operational flexibility, helping them stay ahead of potential adversaries. Ultimately, the framework serves as a powerful platform for those seeking to strengthen their cybersecurity capabilities. However, the approach is not without limitations. Distributed hacking environments face challenges related to process synchronization, resource management, and maintaining global knowledge across nodes. Additionally, achieving fault tolerance and effective error recovery remains difficult in such systems.

Fredj et al. [43] covered the 10 most significant attacks for web applications based on the best-known web vulnerabilities disclosed by the Open Web Application Security Project (OWASP) project. Furthermore, they discussed the key techniques for mitigating web-based threats and the countermeasures commonly employed to address them. Their report presented an in-depth overview of the OWASP Top 10 Web Application Security Risks, offering clear explanations and illustrative scenarios for each category. By mapping out how these risks relate to the broader security landscape, the report underscored the need for organizations to strategically prioritize their defensive efforts. In addition to outlining major vulnerabilities, the report reviewed a variety of security controls and industry best practices for managing web application risks. It serves as a useful reference for organizations aiming to strengthen their understanding of web security and adopt effective protection strategies. Through its detailed discussion of the OWASP Top 10, the report encourages a proactive approach to securing web applications. Additionally, the report highlighted the importance of secure coding principles, the need to educate developers about common risks and mitigation strategies, and the value of employing automated tools to detect weaknesses.

Wibowo et al. [14] stated that web applications are required to respond to the ease of use of Internet technology. Cross-site scripting (XSS) is one of the most widespread security threats or attacks. Numerous solutions can be employed to prevent cyber-attacks, one of which is OWASP Security Shepherd, a secure training and testing platform that offers a wide range of tools and techniques including protection against XSS to strengthen web applications. By combining secure coding practices, automated testing tools, and manual code reviews, OWASP Security Shepherd provides an effective defense mechanism against XSS and other common vulnerabilities. The platform allows both developers and organizations to reduce the risks associated with XSS attacks by ensuring that applications are thoroughly tested and regularly updated. Moreover, OWASP Security Shepherd offers an intuitive interface along with useful features such as user-friendly reports, real-time monitoring, and support for multiple programming languages. These capabilities enable developers to quickly detect, understand, and resolve potential vulnerabilities in their web applications, thereby improving the overall security posture of an organization's online systems.

Muhammet et al. [35] stated that the languages used by web-based systems can cause specific inherent security flaws. Numerous free and paid technologies are available to identify security flaws in online applications. The purpose of this study was to evaluate and analyze the most widely used online vulnerability testing programs. The findings revealed that some tools were significantly more accurate than others. In several cases, the technologies used to detect security flaws proved unreliable, occasionally failing to identify potential vulnerabilities altogether. Despite the varying levels of accuracy, all the examined tools demonstrated the capacity to detect security weaknesses that could expose applications to malicious attacks. These results highlight the need for organizations to rely on multiple vulnerability testing programs rather than a single solution, as no individual tool can effectively detect all types of security issues. This reinforces the importance of employing multiple layers of security controls to achieve comprehensive protection. Furthermore, organizations must ensure that their security tools and testing technologies remain up to date, as newer solutions may provide improved detection capabilities. By using a combination of vulnerability testing programs and maintaining updated security measures, organizations can enhance the resilience of their applications and reduce the risk of successful cyber-attacks.

Wardana et al. [14] declared that vulnerability assessment and penetration testing on published websites following specific standards are critical for information security. Testing was conducted using the NIST 1100-125-125 Standard across the four primary stages of preparation, discovery, attack, and reporting. One high-level, two medium-level and four low-level vulnerabilities were identified. During the penetration testing, it was found that while specific security measures had been implemented, including encryption and authentication techniques, they were not utilized to their full potential. Furthermore, other measures could have been implemented to improve the security of the websites, such as using firewalls and intrusion detection systems. These security measures could have provided more robust protection against potential attacks and alerted the organization to any malicious activity occurring on their websites. This assessment and testing revealed that, despite the presence of specific security measures, there was still a need to enhance security by leveraging all available resources. The testing demonstrated

the need for improving security measures to prevent potential malicious actors from exploiting vulnerabilities. In conclusion, the penetration testing revealed that while security measures were already in place, they needed to be strengthened and utilized more effectively. To this end, recommendations were put forth to ensure that the advantage was taken all the available resources and that the security measures in place were sufficient.

Nagendran et al. [44] described client-side and server-side attacks in classifying web attacks. In addition, they provided an in-depth explanation of how to perform a manual penetration test in web applications to ensure their integrity and security, as well as a guide to testing the OWASP's top 10 security vulnerabilities. The authors also discussed manual web application penetration testing methodologies, which they classified into five phases: reconnaissance, scanning, exploitation, maintaining access and privilege escalation, and clearing tracks and reporting.

Auricchio et al. [45] proposed a design for an automated web application penetration testing architecture. In addition, they developed an orchestration-based approach to web application penetration testing that they called hacking goals. The researchers identified the generic tasks performed during a penetration test and created a mechanism for integrating attacks that implement these tasks into a component that executes them. To enable the orchestration of tasks across all phases of a testing campaign, they also defined a communication protocol between the two elements. To illustrate how the framework could be applied, the researchers demonstrated how multiple types of attacks could be integrated and an ad-hoc behavioral model that can be embedded to detect cross-site scripting attacks.

Alanda et al. [46] performed penetration testing using the black-box method on web applications based on OWASP's most common attack, namely SQL injection. The researchers randomly attacked several commercial, government, and school websites with various SQL injection techniques. Testing was conducted randomly on 10 websites to identify security gaps using SQL injection attacks. According to the testing results, 80% of the websites tested were vulnerable to SQL injection attacks. According to the authors' findings, SQL injection remains the most prevalent threat to web applications. In addition, they provided detailed information about SQL injection and how to prevent it. Alhassan et al. [47] proposed the fuzzy classifier (FC) vulnerability assessment and penetration testing (VAPT) model using the intelligent learning scheme. This model detects vulnerabilities in web applications and indicates the threat or penetration level for recognized cases. The proposed FCVAPT model was evaluated using XSS and SQL injections.

Hasan et al. [30] conducted a literature survey, provided an overview of VAPT, and identified several limitations. They found it beneficial to use VAPT to secure a web application. They also discussed several tools that can be useful for conducting a VAPT process to detect SQL injection, XSS, local file inclusion, and remote file inclusion vulnerabilities. They found that VAPT helped identify security defects very effectively. Albahar et al. [48] compared pen-testing tools for detecting vulnerabilities in web applications based on approved standards and methods to facilitate penetration testers' selection of the most appropriate tools. To enhance the effectiveness of web penetration testers and penetration testers in real-world scenarios, they proposed a benchmarking framework that incorporates the latest research into benchmarking and evaluation criteria, as well as new criteria that provide more comprehensive coverage with benchmarking metrics. Moreover, a score-based comparative analysis was used to evaluate the tool's abilities. They also conducted simulation tests of commercial and non-commercial penetration testing tools. In their study, Burp Suite Professional was rated the highest among the commercial tools, while OWASP ZAP was rated the highest among the non-commercial tools.

Pareek [44] explained the types of penetration testing used for web applications, including white-box, black-box, and gray-box penetration testing. Additionally, they outlined the seven phases of penetration testing for web applications. A review of OWASP's top 10 web application security risks was also conducted. Moreover, they presented five tools for web application penetration testing, namely Astra's Pentest, NMAP, Wireshark, Metasploit, and Burp Suite, in terms of their features and capabilities. Based on the reviewed studies, Table 2 presents the key findings on the penetration test types, as well as the suggested techniques, advantages, and limitations of each study.

Table 2: Summary of related works.

Author	Publication Year	Pen Test Type	Suggested Technique	Advantages	Limitations
Wibowo et al. [42]	2021	Integrated (automated and manual)	An integrated approach for OWASP Security Shepherd, utilizing a combination of secure coding practices, automated tools, and manual code reviews.	OWASP Security Shepherd provides the following: 1. An effective solution for protecting web applications from XSS attacks; 2. An intuitive interface and features such as easy-to-use reports, real-time monitoring, and support for multiple programming languages. This enables developers to quickly identify and address potential vulnerabilities in their web applications, thereby enhancing the overall security of their online assets.	The present web application firewalls only offer basic protection rules that do not consider advancements in the sector. The authors aimed to develop a lightweight and adaptable web application firewall as part of their ongoing project in the future.
Muhammet et al. [16]	2021	Manual	Manual online vulnerability test programs, based on available free and paid technologies, are essential for identifying security flaws in online applications. This is because the languages used by web-based systems may inherently pose certain security risks.	By utilizing multiple vulnerability testing programs, organizations can ensure that their applications remain secure against malicious attackers. Additionally, organizations should ensure that the security measures they use are up to date, as newer technologies may be more effective in detecting security flaws.	The results indicated that the technologies used to detect security flaws were not always reliable and, in some cases, failed to identify any potential vulnerabilities. This underscores the importance of using multiple layers of security measures since a single vulnerability test program may not be sufficient to identify all possible security issues.
Wardana et al. [42]	2022	Manual	Manual penetration testing on published websites follows a standard process with four primary stages: Preparation, Discovery, Attack, and Report.	To improve the security of the websites, other measures could have been implemented, such as using firewalls and intrusion detection systems. These security measures could have provided more robust	The testing demonstrated the need for improved security measures to prevent potential malicious actors from exploiting vulnerabilities. In conclusion, the penetration testing revealed that while security measures

Author	Publication Year	Pen Test Type	Suggested Technique	Advantages	Limitations
				protection against potential attacks and alerted the organization to any malicious activity occurring on their websites.	were already in place, they needed to be strengthened and utilized more effectively.
Nagendran et al. [17]	2019	Manual	Manual web application penetration testing with the following five phases: Reconnaissance, Scanning, Exploitation, maintaining access and privilege escalation, Clearing tracks, and reporting.	Aurichio provided an automated framework for conducting manual penetration tests on web applications.	Performing manual penetration tests requires a great deal of expertise in working with HTTP requests and responses.
Auricchio et al. [18]	2022	Automated	Developed an orchestration-based approach to web application penetration testing called hacking goals.	The work by Alanda et al. presents an automated framework for penetration testing, as the proposed framework is flexible enough to accommodate different attack models, which can be easily customized for various domains.	I found no limitations.
Alanda et al. [19]	2021	Automated and manual	Implemented the black-box method to test web applications for vulnerabilities using OWASP's most common attack, SQL injection.	Various web applications were examined in terms of their penetration methods and the impact of SQL injection. A detailed explanation of SQL injection and its prevention methods was provided.	The conclusion is concise and does not include a discussion of future work.
Albahar et al. [22]	2022	Automated and manual	To enhance the effectiveness of web penetration testers and penetration testers in real-world scenarios, a benchmarking framework was proposed that incorporates the latest research on benchmarking and evaluation.	A comprehensive framework is provided, complete with all the necessary features for penetration testers.	Benchmarking should be applied to other tools, and the framework should be extended to include more new metrics.

Author	Publication Year	Pen Test Type	Suggested Technique	Advantages	Limitations
Ablahd [23]	2023	Automated	A system was proposed that detects web application vulnerabilities using Python 3.7 to identify injection flaws such as command execution and cross-site scripting.	The proposed scanner is easy to use (in each web application) and flexible when it comes to updating.	The authors needed to adapt the scanner to detect other types of web vulnerabilities.
Pareek[24]	2019	Automated	Five web application penetration testing tools were presented namely Astra's Pentest, NMAP, Wireshark, Metasploit, and Burp Suite.	It explains the types, phases, and tools of web penetration testing.	There is not enough discussion about the tools and how to select the optimal one.

3.1. Overview of Web Application Security

Web application security is a critical aspect of modern Internet infrastructure due to the pervasive use of web applications and their susceptibility to cyber-attacks. The diversity of web applications exposes them to numerous security threats, making it essential for organizations to adopt robust security practices. The OWASP (Open Web Application Security Project) Top 10 is a widely recognized framework that highlights the most critical security risks for web applications, providing a valuable resource for developers and security professionals to effectively address these vulnerabilities.

3.2. Importance of Penetration Testing

Penetration testing, also known as ethical hacking, is a crucial method for assessing the security of systems and applications. This process involves simulating attacks to identify and exploit vulnerabilities, thereby providing a realistic evaluation of an application’s security posture. The effectiveness of penetration testing lies in its ability to uncover hidden vulnerabilities that automated tools may miss. By combining manual and automated techniques, penetration testers can thoroughly evaluate security controls and identify potential weaknesses.

3.3. Key Vulnerabilities: SQL Injection and Cross-Site Scripting (XSS)

SQL Injection and Cross-Site Scripting (XSS) are two of the most prevalent and dangerous vulnerabilities affecting web applications. According to the OWASP Top 10, these vulnerabilities are consistently found in web applications. They can lead to severe consequences, including unauthorized access to databases, data breaches, and manipulation of web content[49]. SQL Injection occurs when malicious SQL queries are executed in the database, while XSS allows attackers to inject malicious scripts into web pages viewed by other users. Both vulnerabilities can be effectively identified and mitigated through rigorous penetration testing.

3.4. Advances in Cryptographic Techniques

Discuss the development of advanced cryptographic algorithms using Maple software in their paper "Some Maple algorithms generating diagnostically strong cryptographic examples[49]. "These algorithms are designed to generate strong cryptographic examples that can enhance the security of web applications. The study emphasizes the importance of employing robust cryptographic techniques to safeguard sensitive data and ensure secure communication within web applications. Similarly, it addresses common cryptographic weaknesses in ICT services, particularly in developing countries, in "Some Cryptographic Weakness and the Ways to Avoid Them in ICT Services of Developing Countries[50] ".The paper emphasizes the need to adopt strong cryptographic practices and provides practical recommendations for mitigating these weaknesses. This research highlights the importance of cryptography in securing web applications and mitigating cyberattacks. Additionally, it examines data security

challenges in both the public and private sector sectors, emphasizing effective protection strategies in the digital era[51]. Their study, "Data Security in Public and Private Administration: Challenges, Trends, and Effective Protection in the Era of Digitalization," offers insights into current trends and practical measures for safeguarding data in increasingly digital environments[52].

3.5. Integrating Penetration Testing with Manual, Automated, and Hybrid models testing

The integration of sophisticated penetration testing approaches, has significantly enhanced the ability to identify and exploit vulnerabilities. In today's cybersecurity landscape, penetration testing (pentesting) is a critical component of any robust security strategy. It involves simulating cyberattacks to identify vulnerabilities before malicious actors can exploit them. However, not all pentests are created equal. Organizations can choose from automated, manual, and hybrid approaches, each offering distinct advantages and considerations[53]. Let’s explore these three approaches:

3.5.1. Manual Pentesting

Manual pentesting involves cybersecurity professionals manually probing systems and applications to uncover vulnerabilities. This approach relies on the tester’s expertise, intuition, and creativity.

1. Advantages:
   - Thoroughness: Manual testers can identify complex vulnerabilities that automated tools might miss, providing a deeper and more comprehensive assessment.
   - Contextual Analysis: Human testers can understand the context and potential impact of vulnerabilities, offering more relevant and actionable insights.
   - Adaptability: Manual testing can adapt to new and emerging threats, ensuring a more current and relevant security assessment.
2. Considerations:
   - Time-Consuming: Manual testing is labor-intensive and can take significantly longer than automated testing.
   - Cost: Due to the expertise required, manual testing is generally more expensive.
   - Inconsistency: The quality of manual testing can vary based on the tester’s experience and approach.

3.5.2. Automated Pentesting

Automated pentesting leverages advanced tools and software to scan for vulnerabilities within your systems. These tools simulate attacks and generate reports on identified weaknesses without human intervention.

1. Advantages:
   - Speed and Efficiency: Automated tools can quickly scan large networks and applications, identifying vulnerabilities in a fraction of the time it would take a human tester.
   - Cost-Effective: Automated testing is generally less expensive than manual testing, making it an attractive option for organizations with limited budgets.
   - Consistency: Automated tools follow a predefined set of rules and procedures, ensuring consistent results every time they are run.
2. Considerations:
   - Limited Depth: Automated tools may miss complex vulnerabilities that require human intuition and expertise to identify.
   - False Positives: These tools can generate false positives, identifying issues that are not actual vulnerabilities, which can waste time and resources.

3.5.3. Hybrid Pentesting

Hybrid pentesting combines automated and manual testing to leverage the strengths of both approaches. This method uses automated tools for initial scanning and identification of vulnerabilities, followed by manual testing to delve deeper into the findings.

1. Advantages:
    - Balanced Approach: Hybrid testing offers a comprehensive assessment by combining the speed and efficiency of automated tools with the depth and context provided by manual testing.
    - Cost-Effective and Thorough: While more expensive than purely automated testing, hybrid testing is often more affordable than extensive manual testing, providing a good balance between cost and thoroughness.
    - Flexibility: This approach can be tailored to fit the specific needs and risk profile of the organization, offering a customizable solution.
2. Considerations:
    - Complexity: Managing a hybrid testing approach requires coordination and expertise to ensure that both automated and manual elements are effectively integrated.
    - Resource Intensive: While more cost-effective than full manual testing, hybrid testing still requires significant resources and expertise.

3.6. Continuous Improvement in Web Application Security

Adequate web application security requires continuous improvement and proactive measures to ensure optimal protection. Regular security assessments, timely updates, and ongoing employee training are essential for maintaining a strong security posture. Organizations must remain informed about emerging threats and consistently adopt best practices to safeguard their web applications. Developing comprehensive checklists and security guidelines based on penetration testing results enables organizations to systematically identify, prioritize, and mitigate vulnerabilities, thereby strengthening their overall security framework. The reviewed literature underscores the critical importance of web application security and highlights the central role penetration testing plays in uncovering and addressing security weaknesses. The OWASP Top 10 framework offers a valuable reference point for understanding and mitigating common risks, while advancements in cryptographic techniques provide additional layers of protection. Furthermore, real-world case studies of major data breaches demonstrate the severe consequences of inadequate security practices and reinforce the necessity of continuous improvement. Through integrating automated scanning tools with manual penetration testing and adopting industry best practices, organizations can significantly enhance the security of their web applications and reduce the likelihood of cyber-attacks. This study seeks to contribute to the broader field of web application security by offering practical recommendations and insights grounded in comprehensive analysis and penetration testing.

4. Methodology

The methodology for this research involves both theoretical and practical components. The theoretical component includes an extensive review of existing literature on web application security, penetration testing techniques, and the OWASP Top 10 vulnerabilities. The practical component involves conducting penetration tests on a selected web application to identify and mitigate vulnerabilities.

Nevertheless, practical experiment was conducted in a controlled security-testing environment using an Ubuntu 22.04 LTS virtual machine, where intentionally vulnerable benchmark applications OWASP Juice Shop, and bWAPP (buggy web application) were deployed using Docker and VirtualBox. Five penetration testing tools (OWASP ZAP, Burp Suite, Arachni, Nikto, and Metasploit) were executed independently against the target applications to avoid scan interference, and each tool was tested against forty predefined test cases covering SQL Injection, Cross-Site Scripting (XSS), broken authentication, IDOR, security misconfigurations, and key vulnerabilities from the OWASP Top 10. Scan results were evaluated using standardized metrics including detection accuracy, false positives, scan time, vulnerability coverage, and usability. Automated findings were manually validated using Burp Repeater, browser developer tools, and controlled payload execution to confirm exploitability and eliminate false reports. Each application was reset to its initial state before every scan to ensure reproducibility, and all logs were cross checked against the known vulnerability set of the benchmark applications.

5. Analysis and Discussion

This study advances current knowledge and practice in web penetration testing by demonstrating that no single tool provides comprehensive vulnerability coverage and that effectiveness varies significantly across scanning engines. Our results show that combining multiple tools rather than relying on a single platform substantially improves the breadth and depth of vulnerability detection. This finding contributes to practice by offering a practical, evidence-based framework for selecting appropriate testing tools based on organizational needs. Furthermore, the study identifies the functional strengths and limitations of each tool, providing actionable criteria for tool selection. These include: the ability to both detect and exploit vulnerabilities, the quality and completeness of generated reports, cross-platform compatibility, and support for testing across diverse environments. By outlining these measurable selection factors, the study equips practitioners and organizations with a more structured approach to choosing and deploying penetration testing tools. The results enhance existing knowledge by reinforcing the importance of multi-tool, multi-layer testing strategies, while also offering practical guidance that supports more secure and informed penetration-testing practices in real world settings.

5.1. SQL Injection and Cross-Site Scripting

This section describes SQLI and XSS Web-application vulnerabilities and illustrates attacks that exploit them. SQL Injection. A SQLI vulnerability results from the application's use of user input in constructing database statements. The attacker invokes the application, passing as an input a (partial) SQL statement, which the application executes. This permits the attacker to get unauthorized access to, or to damage, the data stored in a database. To prevent this attack, applications need to sanitize input values that are used in constructing SQL statements, or else reject potentially dangerous inputs.

1. First-order XSS
   A first-order XSS (also known as Type 1, or reflected, XSS) vulnerability results from the application inserting part of the user's input in the next HTML page that it renders. The attacker uses social engineering to convince a victim to click on a (disguised) URL that contains malicious HTML/JavaScript code. The user's browser then displays HTML and executes JavaScript that was part of the attacker-crafted malicious URL. This can result in stealing of browser cookies and other sensitive user data. To prevent first-order XSS attacks, users need to check link anchors before clicking on them, and applications need to reject or modify input values that may contain script code.
2. Second-order XSS
   A second-order XSS (also known as persistent, stored, or Type 2 XSS) vulnerability results from the application storing (part of) the attacker's input in a database, and then later inserting it in an HTML page that is displayed to multiple victim users (e.g., in an online bulletin board application). It is harder to prevent second-order XSS than first-order XSS, because applications need to reject or sanitize input values that may contain script code and are displayed in HTML output, and need to use different techniques to reject or sanitize input values that may contain SQL code and are used in database commands. Furthermore, Second-order XSS is much more damaging than first-order XSS, for two reasons: (a) social engineering is not required (the attacker can directly supply the malicious input without tricking users into clicking on a URL), and (b) a single malicious script planted once into a database executes on the browsers of many victim users.

5.1.1. Example PHP/MySQL Application

PHP is a server-side scripting language widely used in creating Web applications. The program in Fig. 1 implements a simple message board that allows users to read and post messages, which are stored in a MySQL database. To use the message board, users of the program fill an HTML form (not shown here) that communicates the inputs to the server via a specially formatted URL, e.g.,

```
http://www.mysite.com/?mode=display&topicid=1
```

Input parameters passed inside the URL are available in the $ GET associative array. In this example URL, the input has two key-value pairs: mode=display and topicid=1.

Algorithm 1: Insecure PHP Message Board with Database Storage

```
1  // exit if parameter 'mode' is not provided
2  if(!isset($_GET['mode'])){
3  |  exit;
4  }
5
6  if($_GET['mode'] == "add")
7  |  addMessageForTopic();
8  else if($_GET['mode'] == "display")
9  |  displayAllMessagesForTopic();
10 else
11 |  exit;
12
13 function addMessageForTopic(){
14 |  if(!isset($_GET['msg']) ||
15 |  |  !isset($_GET['topicid']) ||
16 |  |  !isset($_GET['poster'])){
17 |  |  exit;
18 |  }
19 |
20 |  $my_msg = $_GET['msg'];
21 |  $my_topicid = $_GET['topicid'];
22 |  $my_poster = $_GET['poster'];
23 |
24 |  // construct SQL statement
25 |  $sqlstmt = "INSERT INTO messages VALUES('$my_msg','$my_topicid')";
26 |
27 |  // store message in database
28 |  $result = mysql_query($sqlstmt);
29 |  echo "Thank you $my_poster for using the message board";
30 }
31
32 function displayAllMessagesForTopic(){
33 |  if(!isset($_GET['topicid'])){
34 |  |  exit;
35 |  }
36 |
37 |  $my_topicid = $_GET['topicid'];
38 |
39 |  $sqlstmt = "SELECT msg FROM messages WHERE topicid='$my_topicid'";
40 |  $result = mysql_query($sqlstmt);
41 |
42 |  //display all messages
43 |  while($row = mysql_fetch_assoc($result)){
44 |  |  echo "Message " . $row['msg'];
45 |  }
46 }
```

Algorithm 1 is a PHP program that implements a simple message board using a MySQL database. This program is vulnerable to SQL injection and cross-site scripting attacks.

This program can operate in two modes: posting a message or displaying all messages for a given topic. When posting a message, the program constructs and submits the SQL statement to store the message in the database (lines 25 and 28) and then displays a confirmation message (line 29). In the displaying mode, the program retrieves and

displays messages for the given topic (lines 39, 40, and 44). This program is vulnerable to the following attacks, all of which our technique can automatically generate:

A. SQL injection attack

Both database queries, in lines 28 and 40, are vulnerable but we discuss only the latter, which exploits the lack of input validation for topicid. Consider the following string passed as the value for input parameter topicid:

- 1′ OR ‘1’=‘1

This string leads to an attack because the query that the program submits to the database in line 40,

- SELECT msg FROM messages WHERE topicid=‘1’ OR ‘1’=‘1’

contains a tautology in the WHERE clause and will retrieve all messages, possibly leaking private information. To exploit the vulnerability, the attacker must create an attack vector, i.e., the full set of inputs that make the program follow the exact path to the vulnerable mysql query call and execute the attack query. In our example, the attack vector must contain at least parameters mode and topicid set to appropriate values. For example:

- mode → display, topicid → 1′ OR ‘1’=‘

First-order XSS attack. This attack exploits the lack of validation of the input parameter poster. After storing a message, the program displays a confirmation note (line 29) using the local variable my poster, whose value is derived directly from the input parameter poster. Here is an attack vector that, when executed, opens a popup window on the user’s computer:

- mode → add
- topicid → 1
- msg → Hello
- poster → Villain<script>alert("XSS")</script>

This particular popup is innocuous; however, it demonstrates the attacker’s ability to execute script code in the victim’s browser (with access to the victim’s session data and permissions). A real attack might, for example, send the victim’s browser credentials to the attacker.

B. Second-order XSS attack

This attack exploits the lack of SQL validation of parameter msg when storing messages in the database (line 25) and the lack of HTML validation when displaying messages (line 44). The attacker can use the following attack vector to store the malicious script in the application’s database.

- mode → add
- topicid → 1
- msg → Hello<script>alert("XSS")</script>
- poster → Villain

Now every user whose browser displays messages in topic 1 gets an unwanted popup. For example, executing the following innocuous input results in an attack:

- mode → display
- topicid → 1

Across the examined applications, SQL Injection (SQLi) and Cross-Site Scripting (XSS) emerged as the most frequently detected vulnerabilities. This aligns with recent studies that consistently rank SQLi and XSS among the most persistent and dangerous threats identified by automated scanners and penetration testers[54], [55].

Our findings reaffirm that despite the availability of mature web frameworks, improper input validation and insecure data-handling practices continue to expose applications to high-severity risks. The recurrence of these vulnerabilities highlights a systemic challenge in secure software development practices, particularly in rapidly built or poorly maintained systems.

Automated tools such as OWASP ZAP and Arachni detected a high proportion of injection-related vulnerabilities, yet they struggled to identify logic-based weaknesses an issue also noted by Khaled[56]. This supports the position that automated scanning alone is insufficient for comprehensive vulnerability assessment.

5.2. Web App Vulnerabilities

5.2.1. The Evolution of Web App Vulnerabilities

FORM components, such as buttons and text fields, are commonly used by web programs to communicate with users. Moreover, the GET or POST variables play a crucial role in the communication process. GET and POST variables are part of the Hypertext Transfer Protocol (HTTP), which facilitates communication between web programs and users by providing them with specific form components, such as text fields and buttons. Improper handling of data items within HTTP requests leads to the most severe security vulnerabilities in web applications. SSL does not eliminate security issues, as it enables secure data transport but does not evaluate HTTP queries. Web apps serve as a gateway to databases that contain crucial application data and assets. Some of the key dangers to the database server layer are SQL injection, illegal server access, and password-cracking attacks. Most SQL injection vulnerabilities are triggered by insufficient input validation. Most online applications store sensitive data in databases or file systems. Developers routinely make errors in the encryption approaches they use to secure this information. Because HTTP is a stateless protocol, web programs use different methods to maintain the session state. A session is a sequence of interactions between the user and the web app during a single visit to the website. A unique string, often referred to as a session ID, is used for session management and is sent to the web server with each request. Most web programming languages enable sessions using GET variables and/or cookies. If an attacker can guess or steal a session ID, they can modify the session of another user[43].

A. Plugin Vulnerabilities

Plugins used in web applications, especially on platforms like WordPress, can introduce significant security vulnerabilities if not properly managed. A standard security issue related to plugins is outdated software. Many plugins receive regular updates from their developers to address security flaws and bugs. Failure to update plugins promptly can leave websites vulnerable to known exploits.

B. Zero-Day Vulnerabilities

Zero-day vulnerabilities are flaws that have been discovered but not yet patched, presenting a significant risk to the security of systems and devices. To mitigate the risks associated with such vulnerabilities, it is crucial to implement firewalls, keep antivirus software up to date, and conduct periodic vulnerability scans.

C. Open-Source Vulnerabilities

Open-source software poses risk due to its transparency, as the code is accessible to everyone, including attackers. Conducting regular security scans and staying informed about the latest security developments are essential for mitigating these risks.

D. SQL Injection and Cross-Site Scripting (XSS)

SQL Injection occurs when malicious SQL queries are executed in the database, potentially allowing attackers to access, modify, or delete data without authorization. This vulnerability poses a significant risk due to its potential impact on data integrity and confidentiality. Cross-site scripting (XSS) enables attackers to inject malicious scripts into web pages that other users view, allowing them to compromise the security of these pages. These scripts can steal sensitive information, manipulate content, or perform actions on behalf of the user without their consent.

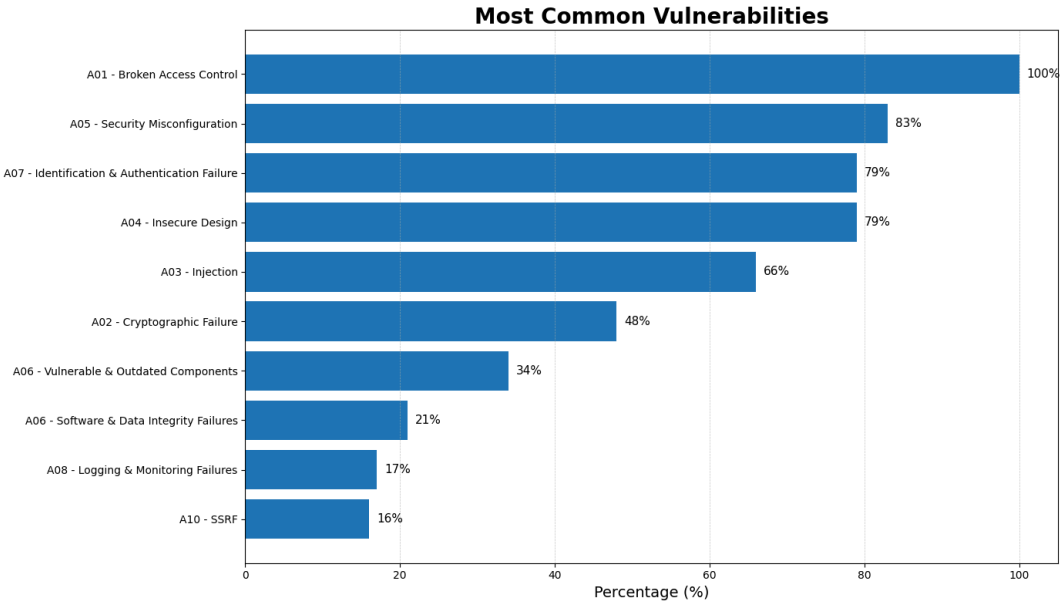

Fig. 1: Top 10 security threats for web applications.

E. Cross-Site Request Forgery (CSRF)

According to OWASP, CSRF attacks occur when a malicious actor forces a user to perform unwanted actions without their knowledge of a web application where they are authenticated. These attacks often exploit the trust between the user's browser and the targeted web application. An effective countermeasure to prevent CSRF attacks is the use of a "challenge token," also known as a "CSRF token."

F. Top 10 Security Threats for the Web Environment

OWASP has produced a renowned top-10 list that includes the most critical security vulnerabilities in web applications and provides guidance on how to address these issues. Fig. 1 lists the top 10 OWASP vulnerabilities.

In the following, we provide an investigation of the top 10 security threats for web environments. The investigation encompasses a broad range of topics, including malicious software, unpatched vulnerabilities, inadequate network and server security practices, insecure user authentication protocols, and other security risks.

G. Broken Access Control

Some web applications verify access permissions at the function level before making the functionality available to the user. Nonetheless, once each feature is accessible, programs must pass the same access control check as the server. When requests are not verified, attackers can get access to features without the necessary authorization. The following are examples of attacks that can exploit the broken access control flaw. In a local file inclusion attack, the attacker attempts to locate a page that accepts a path to a file as input, which will be included in the calling page. Moreover, the distant file inclusion attack is similar to the local file inclusion attack, except that instead of including files on the same server, the attacker manipulates user input to include remote files[14], [57].

H. Cryptographic Failure

Cryptography refers to the methods and procedures used to maintain secrecy, non-repudiation, integrity, and authenticity. A cryptographic failure is a significant online application security issue that exposes sensitive application data due to the use of a poor or non-existent cryptographic method. These data can include passwords, patient health details, company secrets, credit card information, email addresses, and other sensitive user information. Modern online applications process data both at rest and in transit, necessitating sophisticated security procedures for full threat mitigation.

The following are examples of attacks that can exploit the cryptographic failure flaw. Some deployments utilize weak cryptographic algorithms that can be cracked within a reasonable timeframe. Cryptographic failures

include the transmission of secret material in plain text, the use of outdated or insecure algorithms, as well as the exploitation of side-channel information or cryptographic error signals. Inadequate randomness for cryptographic functions and the presence of sensitive data in source control are among the common causes of these failures[58].

I. Injection

An interpreter may receive or be sent untrusted information from an attacker. The attacker can trick the interpreter and trigger unauthorized instructions by supplying malicious information. The following three types of injection attacks are the most serious: SQL injection, code injection, and XPath injection.

The following are examples of attacks that can exploit the injection flaw. The first category of attack is known as an SQL injection attack, which involves injecting SQL instructions into input forms or queries to access databases or modify their contents, such as by deleting or altering database information. The second category of attack is code injection, which involves injecting code that the application understands and executes to exploit the sloppy handling of untrusted input. The third category of attack is called XPath injection, which occurs when a web application constructs an XPath query for XML data using user input [14], [57].

J. Insecure Design

To avoid security vulnerabilities, developers are encouraged to apply secure design patterns, conduct thorough threat modeling, and utilize well-established reference architectures throughout the application development process. Insecure design flaws arise when developers, quality assurance teams, or security teams fail to anticipate and assess potential threats during the design phase. Such flaws often result from neglecting established security best practices and overlooking how design decisions may be exploited by attackers. Various attack vectors can take advantage of insecure design, underscoring the need for proactive security planning. As the threat landscape continues to evolve, ongoing and iterative threat modeling is essential to ensure that systems remain protected against both known and emerging attack methods. It is challenging to identify and rectify architectural flaws, such as unprotected credential storage, trust boundary violations, the generation of error messages containing sensitive information, and improper isolation or compartmentalization without a secure design [59].

K. Security Misconfiguration

A security misconfiguration problem occurs when one or more components of the system, such as applications, frameworks, application servers, web servers, database servers, network routers, and platforms, are not correctly configured. Secure settings must be developed, implemented, and consistently maintained to ensure optimal security. The following are examples of attacks that can exploit the security misconfiguration flaw. Default settings are frequently the source of such danger [60]. The attacker might use this issue to launch several attacks. The degree and location of the misconfiguration determine the intensity of the attack.

L. Vulnerable and Outdated Components

A software component such as a module, package, library, or API-forms part of a larger system and contributes to the overall functionality of an application. However, these components can introduce significant security risks when they are outdated, unsupported, or contain known vulnerabilities. Component-based vulnerabilities occur when insecure or obsolete components are unknowingly deployed in production environments. For example, an organization may install widely used software like OpenSSL but fail to apply security updates or patches after new weaknesses are disclosed. Since many software components operate with the same privileges as the hosting application, any flaw within the component becomes a direct threat to the entire system. Using components with known vulnerabilities exposes an application to a wide range of attacks across the application stack. Common attack scenarios include code injection, buffer overflow, command injection, cross-site scripting (XSS), and the exploitation of outdated libraries or frameworks. In one such scenario, an attacker may leverage an unpatched component to execute malicious code on the server, compromising the integrity, confidentiality, and availability of the application. The attacker gains access to a company's internal network and then utilizes a scanning tool to

identify internal systems with unpatched or outdated components. The attacker then exploits a vulnerability in the obsolete component to install malicious code on the application server [61].

M. Identification and Authentication Failures

Hackers exploit this vulnerability to take advantage of improper authentication, as its name suggests. A hacker can access user information, passwords, ID sessions, and other login credentials, thereby posing a significant security risk [62]. The following is an example of attacks that can exploit the identification and authentication failure flaw. Credential stuffing is considered a type of broken authentication attack that is driven by brute force.

N. Software and Data Integrity Failures

Failures of software and data integrity are caused by code and infrastructure that do not protect against integrity violations. The following are examples of attacks that can exploit software and data integrity flaws. A good example is when an application depends on plugins, libraries, or modules downloaded from untrustworthy sources, repositories, or content delivery networks. CI/CD pipelines that are insecure can expose systems to unauthorized access, malicious code, and system compromises. Furthermore, many applications now allow auto-updates, whereby updates are downloaded without enough integrity verification and applied to previously trusted applications. Attackers could upload their updates to all installations and distribute them [63].

O. Security Logging and Monitoring Failures

Without logging, suspicious actions and events can remain unmonitored for more extended periods, potentially allowing security breaches to continue undetected for longer than they would with more effective logging. Website hackers can cause significant damage, but hacking is becoming increasingly difficult if web application owners fail to monitor code behavior for suspicious activity. Monitoring systems can be beneficial in this situation. The following is an example of attacks that can exploit the security logging and monitoring failures flaw. Cyberattacks can have repercussions that leave one with a limited understanding of what has happened to their system if they lack a proper logging and monitoring process [64].

P. Server-Side Request Forgery (SSRF)

An attacker can exploit this vulnerability to send requests to an unintended location via a server-side application. The following are examples of attacks that can exploit the SSRF flaw. In a typical SSRF attack, an attacker might also use SSRF to connect the server to internal-only services within an organization's infrastructure. It is also possible for them to force the server to connect to arbitrary external systems, exposing credentials and sensitive data [65].

5.3. Web Penetration Test

Web security is an important concern as the Internet expands and web applications are increasingly used in different fields, including the military, health care, and finance. Web security is ensured by penetration tests. Manual or automatic penetration tests can be conducted.

5.3.1. Web Penetration Testing Tools

This section aims to present attack tools that can be utilized to perform penetration testing based on the type of vulnerability present in the web environment. Moreover, it provides an overview of web penetration testing tools. Table 3 shows the list of web vulnerability and a corresponding tool we can use to detect it.

5.3.2. Overview of Penetration Testing Tools

Seven commercial and open-source testing tools are covered in this section. These are Netsparker, Acunetix, Vega, OWASP ZAP, Wapiti, IronWASP, and W3af. Each tool has unique features and advantages that can be used to identify a variety of web application security vulnerabilities.

Table 3: Web vulnerabilities and attack tools.

Web Vulnerability	**Attack tool**
Carriage return and line feed (CRLF) injection	RLF-Injection scanner
Components with known vulnerabilities	Vulners API
Cross-origin resource sharing (CORS) policy	CORScanner
Cross-site scripting (XSS)	XSSMap
Injection flaw	Custom
Directory traversal	LFISuite
HTTP response splitting	Custom
HTTP verb tampering	nmap HTTP-methods script
Improper certificate validation MassBleed	MassBleed
Insufficient transport layer protection	Custom
Lightweight directory access protocol (LDAP) injection	Custom
Improper certificate validation	MassBleed
Insufficient transport layer protection	Custom
Lightweight directory access protocol (LDAP) injection	Custom
Operating system (OS) command injection	Commix
Remote file inclusion (RFI)	Fimap
SQL injection SQLmap	XML
External entities (XXE)	Custom

A. Netsparker

Netsparker is an online security testing tool. It detects and discloses security flaws at the application level of any website. Netsparker comes in two flavors: desktop and cloud. We can scan hundreds of websites or web-based apps at the same time using the cloud version[36]. A desktop version is a convenient tool that can be used on individual websites, while the cloud version enables users to scan multiple websites simultaneously, making it an incredibly powerful tool for website administrators and developers.

B. Netsparker

Acunetix is an online security testing tool that comprehensively monitors and regulates websites, particularly those dependent on HTML and JavaScript. The software development lifecycle interfaces with project management or bug-tracking systems and contains extensive compliance reports. It runs independently of the operating system by using web browsers[66]. All one needs to do is enter the URL of the target website, and it comes with all the necessary features. Acunetix is the ideal tool for monitoring and regulating websites, especially those that are heavily reliant on HTML and JavaScript.

C. Vega

Vega is a free and open-source online security testing tool for detecting flaws in web applications, and its graphical user interface (GUI) is built in Java. Vega has two points of view, which are scanner and proxy. For debugging, the Vega interactive web app provides a blocking proxy. The attack modules for Vega are written in JavaScript, and because these are open source, they may be enhanced via a JavaScript API and modified by the user[67]. Vega is a very powerful tool for debugging web applications, since it is capable of identifying security flaws that are hidden from the user. It also offers great flexibility, allowing the user to customize their attack scenarios by adding new attack modules and modifying existing ones.

D. OWASP ZAP

OWASP is a multinational non-profit organization dedicated to improving software security. ZAP is a simple, open-source integrated penetration testing tool for discovering vulnerabilities in online applications. OWASP openly

distributes papers, methodologies, documentation, and tools on the subject of web app security[68]. Utilizing the security tools provided by OWASP, such as ZAP, and following its methodologies for secure coding are essential for organizations when building or maintaining applications. In addition to providing security tools such as ZAP, OWASP also offers educational resources for those involved in the process of building or maintaining software.

E. Wapiti

Wapiti is a free online security testing tool for detecting flaws in web applications. It runs a black-box test, which means it does not examine the application's source code but instead scans the web pages of the web application being tested and searches for scripts and forms that potentially inject data. Wapiti functions as a fuzzier after it has a list of URLs, forms, and their inputs, injecting payloads to test if a script is susceptible[69]. This can be used to detect common issues, such as SQL injection, XSS, local and remote file inclusion, LDAP injection, and server-side request forgery. Wapiti can also detect different kinds of vulnerabilities in an application, such as weaknesses in authentication systems, improper error handling, and weak encryption functions.

F. IronWASP

Iron Web Application Advanced Security Testing Platform (IronWASP) is a free and open-source web security testing tool for detecting flaws in web applications. It can identify more than 25 web vulnerabilities. It is a GUI-based utility created in Python and Ruby and can identify false positives and false negatives. IronWASP generates HTML and RTF reports. It can be supplemented using plug-ins or modules written in Python, Ruby, C#, or VB.NET[70]. IronWASP is easy to use and set up and includes a range of tools such as fizzers, proxies, crawlers, traffic analyzers, and even site map generation tools. It is highly versatile, and its modularity makes it an ideal tool for penetration testing, allowing users to combine different features and create powerful solutions tailored to their own needs.

G. W3af

W3af is a free, open-source tool for automating the scanning of web applications. Both GUI and command line interfaces are available for this tool, which can assess a web application for vulnerabilities and exploit them. There are interconnected plugins that share information between them[45]. This allows W3af to crawl a web application, map its contents, detect known vulnerabilities, and identify problems that may arise from the application's source code. It is important to note that W3af should only be used by experienced professionals, as its powerful capabilities can lead to system damage if used incorrectly. W3af offers users the ability to customize and fine-tune an application according to their specific needs.

In Table 4, we use a set of metrics in terms of the technology being used, the programming language used during tool development, the supported platform on which the tool can be used, the supported interface on which the tool was developed, the online or offline status during tool use, the vulnerabilities detected through this tool, tool usability, and tool cost. These metrics will be useful for the relevant decision-makers during the tool selection process.

5.4. Strengths and Limitations of Tools

As shown in Table 5, the comparative results indicate that no single tool provides full vulnerability coverage. Burp Suite demonstrates the highest detection performance for complex injection and authentication-related flaws, while OWASP ZAP offers accessible automated scanning suitable for continuous testing pipelines. Arachni excels in XSS detection due to its DOM analysis capabilities but lacks recent maintenance. Nikto remains useful for rapid server-level assessments but performs poorly with application logic flaws. Metasploit provides advanced exploitation capabilities but is not optimized for initial vulnerability discovery. These findings confirm that combining tools maximizes detection breadth and reduces false negatives.

5.5. Managerial Implications

Web applications are becoming more prevalent in corporate, public, and government services today as a result of advances in web technologies and a changing business environment. While web applications can make life easier and more efficient, several security threats could pose significant risks to an organization's IT infrastructure if they

Table 4: Penetration testing tools.

Tool	Technology	Platform	Interface	Online or offline	Vulnerabilities	Usability	Cost
Netsparker	PHP, Java	Web	Command line interface	Online	Identify vulnerabilities such as heartbleed SSL in web applications.	Setup and use are extremely simple.	Request a quote from Sales
Acunetix	Java	Uses web browsers to run independently of the operating system	GUI	Online	More than 4500 vulnerabilities	Easy-to-use and intuitive	Request a quote from Sales
Vega	Java	Linux, OS X, and Windows	GUI	Online	Identify vulnerabilities such as reflected cross-site scripting, stored cross-site scripting, blind SQL injection, remote file inclusion, shell injection, and more.	Easy to use.	Free
Wapiti	Python	Unix/Linux, FreeBSD Mac OS, OSX, Windows	GUI	Online	More than 23 vulnerabilities	Easy and fast activation and deactivation of attack modules.	Free
OWASP ZAP	Java	Linux, Mac OS, OSX, Windows	GUI	Online	Examines the web application for issues linked to SQL injection. Authentication failure. Exposed sensitive info. Compromised access control. Misconfiguration of security. XSS deserialization is insecure. Components that have known flaws.	Easy to use and report vulnerabilities.	Free
IronWASP	Python and Ruby	Linux, Mac OS, OSX, Windows	Both the GUI and command line interfaces.	Online	More than 25 web vulnerabilities	Beginners may utilize it, since it is extremely simple to use.	Free
W3af	Python	Linux, Mac OS, OSX, Windows	Both the GUI and command line interfaces.	Online	Identify vulnerabilities such as SQL injection, cross-site scripting, guessable credentials, unhandled application problems, and PHP misconfigurations.	Fairly simple to install, and the automatic SVN updates will assist both users and writers in resolving problems rapidly.	Free

Table 5: Comparative analysis of common web penetration testing tools.

Tool	Detection Accuracy	Automation Level	Speed	Usability / Learning Curve	Strengths	Limitations
OWASP ZAP	Medium–High	High	Fast	Easy	Free, good for automated scanning, strong community support	Misses some logic/DOM-based vulnerabilities
Burp Suite (Pro)	High	Medium–High	Medium	Moderate	Best-in-class interceptor & scanner, finds complex vulnerabilities	Paid tool, requires expertise
Arachni	Medium	High	Fast	Moderate	Excellent for XSS detection, distributed scanning	Project no longer actively updated
Nikto	Low–Medium	High	Very Fast	Easy	Good for server misconfigurations	Weak in application-layer vulnerability detection
Metasploit	High (Exploitation)	Medium	Medium	Difficult	Strong exploitation & payload testing	Not primarily a scanner; needs expert user

are not handled correctly. Since attacks are now specifically targeting security flaws in web application designs, the traditional network security measures and technologies may no longer be adequate for safeguarding web applications from new threats. Along with the development of web applications, new security measures, both technical and administrative, should be implemented. Since the number of web applications continues to increase, it is important to know what threats exist for web applications. An organization can be targeted by a web application threat through its website or applications. During the development process, organizations should address these security concerns by implementing penetration testing to find web vulnerabilities and secure them before real attackers can exploit them. In this paper, we aim to help organizations understand web application threats, mitigate them, and choose the best tool for performing web penetration testing.

5.6. Practical and Social Implications

Web applications are becoming increasingly vulnerable to cyber-attacks as the internet grows and evolves. The importance of testing one's web application's cyber security has never been greater. This article covered some of the best web application penetration testing tools to ensure that individuals and organizations can test their applications' security and ensure they will withstand any attacks.

6. Protection Mechanisms

Installed during the deployment phase and capable of offering immediate security assurance, protection mechanisms are the most widely adopted solution for Web application security. However, though protective technologies such as antivirus software, network firewalls, and IDSs (intrusion detection systems) offer comparatively secure protection at the host and network levels, application-level [71] protection technologies are still in their infancy. Park and Sandhu's cookie-securing mechanism can be adopted to eliminate XSS, but it requires explicit modifications to existing Web applications. Scott and Sharp [72], [73] proposed the use of a gateway that filters invalid and malicious inputs at the application level; Sanctum's AppShield [74], Kavado's InterDo [75], and several commercial products now offer similar strategies. Most of the leading firewall vendors are also using deep packet inspection [76] technologies in their attempts to filter application-level traffic. According to a recent Gartner report, those that don't offer application-level protection will eventually "face extinction." Although application-level firewalls offer immediate assurance of Web application security, they have at least three drawbacks: a) they require careful configuration [77], b) they blindly protect against unpredicted behavior without investigating the actual defects that compromise quality, and c) they induce runtime overhead.

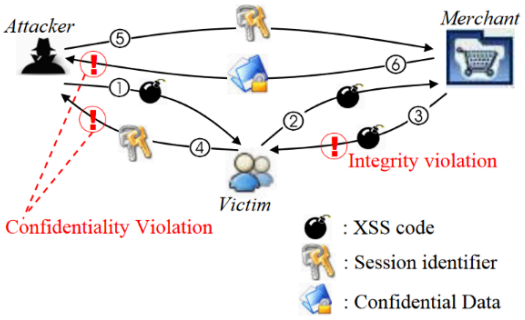

Fig. 2: Web application vulnerabilities result from insecure information flow, as illustrated using XSS.

6.1. Formalizing Web Application Vulnerabilities for Testing and Verification

Adopted during the development phase, software testing and verification are two established technologies for improving software quality. Though incapable of offering immediate security assurance, the two technologies can assess software quality and identify defects. To understand how they can be applied to Web applications, we have to first formally model Web application vulnerabilities. The primary objectives of information security systems are to protect confidentiality, integrity, and availability [78]. From the examples described in Section 1, it is obvious that for Web applications, compromises in integrity are the leading causes of compromises in confidentiality and availability. The relationship is illustrated in Fig. 2. When untrusted data is used to construct trusted output without proper sanitization, data integrity violations occur, leading to escalations in access rights that result in compromises to availability and confidentiality.

Both software testing and verification techniques can be used to identify illegal information flow, specifically to identify violations of Web application noninterference policies [79]. We first make the following assumptions:

1. Assumption 1: All data sent by Web clients in the form of HTTP requests should be considered untrustworthy.
2. Assumption 2: All data local to a Web application are secure.
3. Assumption 3: Tainted data can be made secure with appropriate processing.

Based on these assumptions, we then define the following security policies:

1. Policy 1: Tainted data must not be used in HTTP response construction.
2. Policy 2: Tainted data must not be written into local Web application storage.
3. Policy 3: Tainted data must not be used in system command construction.

Assumption 1 says that all data sent by Web clients (in the form of HTTP requests) should be considered untrustworthy. A majority of Web application security flaws result when this assumption is ignored or neglected. The Web uses a sessionless protocol, in which each URL retrieval is considered an independent TCP session, established when the HTTP request is sent and terminated after the response is retrieved. Many transaction types (e.g., those that support user logins) require session support. To keep track of sessions, Web applications require clients to include a session identifier within an HTTP request. An HTTP request consists of three major components: the requested URL, form variables (also known as parameters), and cookies. In practice, all three are used in different ways to store session information. Cookies are the most frequently used, followed by hidden form variables and URL requests.

To manage sessions, Web applications are written so that browsers include all session information following initial requests that mark the start of a session, and processing HTTP requests entails retrieving that information. Although such information is transferred to the client by the Web application, it should not be considered trustworthy when read back from an HTTP request. The reason is that such information is typically stored without any form of integrity protection (e.g., digital signatures) and is therefore susceptible to tampering. Using such information

```
<script>alert(“WAVES_TEST_1”) ;</script>
```

Fig. 3: An example of our test pattern for XSS.

to construct HTML output without prior sanitization is considered a Policy 1 violation, the most frequent cause of XSS.

Assumption 2 states that all data local to a Web application should be considered secure. This includes all files read from the file system and data retrieved from the database. According to this assumption, all locally retrieved data are considered trusted, resulting in Policy 2, which states that system integrity is considered compromised whenever untrustworthy data is written to local storage. Since most applications use client-supplied data to construct output, our model would be too strict without Assumption 3, which states that untrustworthy data can be made trustworthy (e.g., malicious content can be sanitized and problematic characters can be escaped). XSS vulnerabilities result in Policy 1 or Policy 2 violations. Script injection vulnerabilities such as SQL injection are generally associated with Policy 3 violations.

6.2. Software Testing for Web Application Security

For Web application security, one advantage of software testing over verification is that it considers the runtime behavior of Web applications. It is generally agreed that the massive number of runtime interactions that connect various components is what makes Web application security such a challenging task [72], [80]. Security testing tools for Web applications are commonly referred to as Web Security Scanners (WSS). Commercial WSSs include Sanctum’s AppScan [81], SPI Dynamics’ WebInspect, and Kavado’s ScanDo [75]. Reviews of these tools can be found in [66], but to our best Knowledge, no literature exists on their design. Our contribution in this regard is a security assessment framework, for which we have named the Web Application Vulnerability and Error Scanner, or WAVES. We describe below WSS design challenges and solutions based on our experiences with WAVES.

6.2.1. Output Observation

After submitting a test case to a DEP, its output (HTTP response) is analyzed to detect any Policy 1 violations. To avoid XSS vulnerabilities, client-submitted data containing <script> HTML tags must be processed before being used for output construction. Proper processing entails a) outputting errors that indicate the detection of an attack, b) removing the tag while still processing the request, and c) encoding the <script> tag so that it is displayed rather than interpreted by the browser. To help users determine whether a DEP is taking such sanitization steps, we have designed test patterns that trigger the execution of a special JavaScript by the browser when it renders the DEP output, indicating the absence of a sanitization routine. An example test pattern is shown in Fig. 3.

As described in Section 2.3.6, Microsoft’s Internet Explorer (IE) was added as a core component of WAVES. Accordingly, after submitting the test pattern to a DEP and retrieving its output, it is possible to monitor embedded IE behavior. Suppose IE makes an attempt to display a "WAVES_TEST_1" message box after the response is retrieved. In that case, we know that a) the DEP is using one of its arguments to construct output, and b) it did not perform proper sanitization before output construction. Such DEPs are considered vulnerable to XSS.

6.2.2. Test Case Reduction

For any DEP accepting n arguments, the naïve approach requires n × m test cases for testing against m malicious patterns. To reduce the number of test cases, we modified the test patterns according to the arguments in which the patterns were placed. For example, if placed in the first argument of a DEP, the test pattern shown in Fig. 3 will change to:

<script>alert(“WAVES_TEST_1_ARG_1”);</script>

This allows for the use of IE behavior to identify vulnerable arguments. Using this strategy, we placed modified versions of the same malicious pattern into all arguments of a targeted DEP. This approach requires only $1 \times m = m$ case to be tested against m malicious patterns. When two or more malicious patterns appear in the output, the message box events are captured sequentially, and vulnerable arguments are identified.

Table 6: Hyperlink and navigation techniques.

Method	Description	Example
Traditional HTML Anchors	The most common way to link to other pages or resources.	Google
Framesets	Used to divide the browser window into multiple frames, each loading a separate HTML document.	<frame src="http://www.google.com/top_frame.htm">
Meta Refresh Redirections	An HTML tag that instructs the browser to automatically refresh or redirect to a new URL after a specified time.	<meta http-equiv="refresh" content="0; URL=http://www.google.com">
Client-side Image Maps	Defines clickable areas within an image that act as hyperlinks.	<area shape="rect" href="http://www.google.com">
Javascript Variable Anchors	Uses JavaScript to dynamically generate links, often by concatenating a variable with a URL.	document.write("\ " + LangDir + "\index.htm");
Javascript New Windows and Redirections	Uses JavaScript functions to open new browser windows or redirect the current page.	window.open("\ " + LangDir + "\index.htm"); window.href = "\ " + LangDir + "\index.htm";
Javascript Event-Generated Executions	Executes an action, such as a redirection, in response to a user event like a click or mouseover.	HierMenus (http://www.webreference.com)

6.2.3. Implementation

WAVES' system architecture is shown in Fig. 4. The WebCrawler acts as an interface between Web applications and software testing mechanisms. Without them, we would not be able to apply our testing techniques to Web applications. To make the WebCrawlers exhibit the same behaviors as browsers, they were equipped with IE's Document Object Model (DOM) parser and scripting engine. We chose IE's engines over others (e.g., Gecko[82] from Mozilla) because IE is the target of most attacks. User interactions with Javascript-created dialog boxes, script error pop-ups, security zone transfer warnings, cookie privacy violation warnings, dialog boxes (e.g., "Save As" and "Open With"), and authentication warnings were all logged but suppressed to ensure continuous WebCrawler execution. Note that a subset of the above events is triggered by our test cases or by errors in the web application. An error example is a JavaScript error event produced by a scripting engine during the runtime interpretation of JavaScript code. The WebCrawler suppresses the dialog box triggered by the event and performs the appropriate processing. When an event indicates an error, it logs the event and prepares corresponding entries to generate an assessment report.

In designing the WebCrawler, we examined how HTML pages expose the presence of DEPs and other related pages, and identified the mechanisms listed in Table 6.

A. Form submissions.

We established a sample site to test several commercial and academic web crawlers, including Teleport, WebSphinx [68], Harvest [61], WebGlimpse [69], and Google. None of the evaluated tools were able to crawl beyond the fourth level of revelation approximately half the capability achieved by the WAVES web crawler. WAVES was able to reach revelation levels 5 and 6 because it could interpret JavaScript. Revelation level 7 also involves identifying JavaScript-generated links, but only those triggered by user-driven events such as onClick or onMouseOver. WAVES supports this through an event-generation process that simulates user interactions with active content. This capability allows WAVES to detect malicious components and uncover additional URLs that would otherwise remain hidden. During this stimulation process, JavaScript embedded within the event handlers of dynamic components is executed, potentially exposing new links. Modern web applications frequently use DHTML-based menu systems that reveal navigation options only through dynamic events, meaning many URLs can only be discovered by crawlers capable of handling level-7 revelations. Additionally, although the primary function of the Injection Knowledge Manager (IKM) is to generate variable candidates for bypassing validation mechanisms,

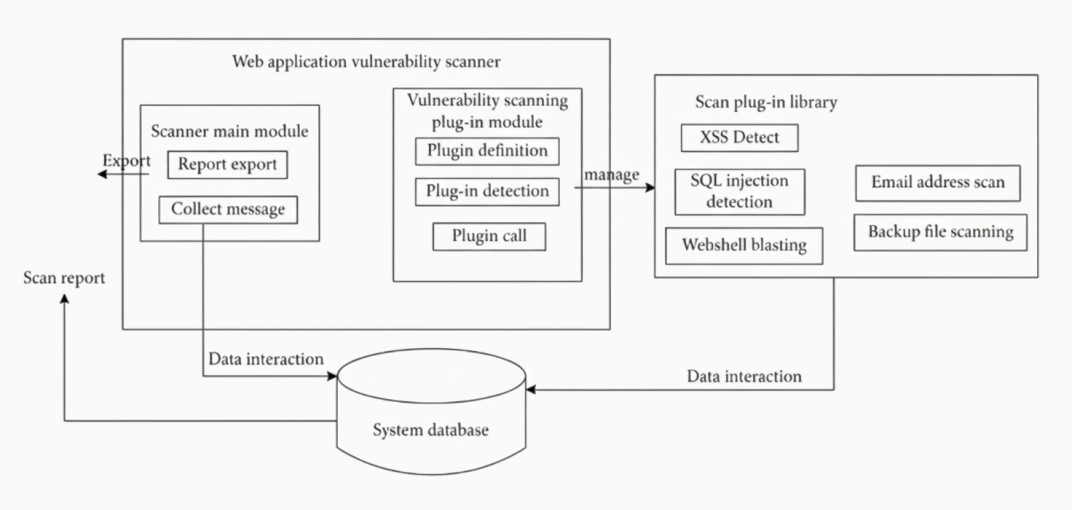

Fig. 4: System architecture of WAVES.

this same knowledge can be leveraged during crawling. When the crawler encounters a form, it queries the IKM for suitable input values. These values are then submitted to the application, enabling deeper page discovery and enhancing the crawler's ability to explore complex web structures.

In the interest of speed, we implemented a URL hash (in-memory) to eliminate disk access during the crawling process. A separate 100-record cache helped to reduce global bottlenecks at the URL hash. See also Cho and Garcia-Molina [85] for a description of a similar implementation strategy. The database feeder does not insert retrieved information into the underlying database until the crawling is complete. The scheduler is responsible for managing a breadth-first crawling of targeted URLs; special care has been taken to prevent web crawlers from inducing harmful impacts on the Web application being tested. The dispatcher directs selected target URLs to the web crawlers and controls crawler activity. Results from crawling and injections are organized in HTML format by the report generator.

6.2.4. Investigate Widely Used WAVS

Web application vulnerability scanning can be classified into three main types. Black-box scanning performs a scan on a given target without accessing the internal source code of the targeted web application. WAVS is an example of this type. White-box scanning, meanwhile, gains complete access to the source code of the web application before it performs the scan. Finally, grey-box scanning offers the penetration tester partial knowledge of the application source code. Information such as hidden path and software version can be supplied [6], [86]. WAVS are available on an open-source and commercial basis [8]. Open-source scanners are free to use and allow users to access (and thus evaluate) their source code. Commonly used open-source WAVS include OWASP ZAP, Wapiti, Vega, W3AF, and Arachni [48], [69]. The main commercial WAVS, meanwhile, are BurpSuite, Acunetix and Netsparker [87], Based on the research findings by Albalawi et al. [88] and the work of Shahid [89] the widely used WAVS that are up to date and recognised for their proficiency in detecting OWASP 2021 vulnerabilities are as follows:

Acunetix is one of the most widely recognised commercial WAVS. It is adept at uncovering an array of vulnerabilities, including but not limited to SQL injections, Cross-Site Scripting (XSS), Host Header Injection, and an extensive list comprising over 3000 web-related vulnerabilities.

- BurpSuite is available in both free and commercial licenses. BurpSuite takes on the role of a Man-In-The Middle (MITM), HTTP requests. This process allows it to capture and analyse requests originating from the target web application server.
- OWASP ZAP, The OWASP Zed Attack Proxy, is one of the most widely recognized open-source WAVS. It was created by the OWASP team.

- Arachni is a free and open-source WAVS. It can navigate complex pathways dictated by a web application's complexity. Arachni excels at identifying vulnerabilities from the OWASP 2021 list, making it a valuable asset in enhancing web application security.
- Nikto is an open-source WAVS. It scans server configuration files. One limitation of Nikto is it primarily focuses on server configuration and may not cover all aspects of web application security, leaving potential vulnerabilities unaddressed.

Alazmi et al. [87] used Arachni and OWASP ZAP to address the effectiveness of WAVS. Given that WAVs used for this project needed to be automated from a framework, an important selection criterion was the ability to run from the command-line interface (CLI). Since both Arachni and OWASP ZAP can start from the CLI, and also since they have both recently been updated, these two WAVs were selected for use within the proposed framework. Based on the sources [90], [91].

Hance et al. [92] suggested a novel attack framework utilising a distributed attack plat-form that incorporated a control scheme for automating vulnerability detection. Qiu et al. [93] developed an automated penetration testing algorithm that exploited vulnerabilities based on a scanning report. Zhou et al. [94] proposed the Network Information Gain Based Auto-mated Attack Planning (NIG-AP) algorithm to automate penetration testing phases that use the reward system. Minh et al. [95], meanwhile, automated vulnerability assessments at the commit level, triggering them with each new commit made to the codebase.

6.2.5. Experimental Result

We evaluated WAVES' DEP discovery ability by comparing its crawling (the number of pages retrieved for a target site) with that of another web crawler. From our tests[96], Teleport proved to be the most thorough among a group of Web Crawlers that included WebSphinx [83], Larbin, and Web-Glimpse [84]. This may be explained by Teleport's incorporation of both HTML tag parsing and regular expression-matching mechanisms, as well as its ability to parse Javascript statically and generate simple form submission patterns for URL discovery. On average, WAVES retrieved 28 percent more pages than Teleport when tested across a total of 14 sites [96]. We attribute the discovery of the extra pages to WAVES' script interpretation and automated form completion capabilities. In the case study evaluating the effectiveness of the proposed scanning modes, the heavy mode identified 80 percent of all errors found through static verification [97]. This demonstrates that our remote, black-box testing approach provides a valuable alternative to static analysis, particularly when source code or local access to the target web application is unavailable. The 58.4% coverage achieved in relaxed mode indicates that effective and non-intrusive testing can still be performed under such constraints. Additionally, the identification of 55 strictly vulnerable sites during a 48-hour relaxed-mode scan supports several conclusions: (a) the proposed mechanism for testing insecure information flow is effective in detecting XSS vulnerabilities, (b) non-destructive testing can still produce practical and actionable results, and (c) XSS remains a persistent and significant threat to modern web applications. Moreover, because tools comparable to WAVES are actively being developed and leveraged by malicious actors, these findings highlight that vulnerable websites can be easily discovered through controlled "attacks" similar to those used in this study though with far more harmful intent.

6.3. The takeaway message from this survey

Security testing for modern web applications is increasingly complex due to factors such as the rise of Rich Internet Applications (RIAs), extensive crawling needs, client-side problem of input validation and output encoding for server responses, lightweight encryption storage, and potential repudiation attacks. However, they also continue to be focused on by a slew of cyber-attackers who have been able to take advantage of various persistent web application vulnerabilities, such as SQL injection and cross-site scripting (XSS) or misconfigurations. Manual penetration testing provides a comprehensive reality check, but is expensive, while the speed and scale of automated tools may miss subtle threats, implying a hybrid model is necessary. Organizations need to follow an integrated approach enabled by frameworks such as the OWASP Top 10, strengthened by robust cryptographic practices, and the imposition of strong security policies for secure coding in a continuous testing manner. Such as tooling selec-

tion, periodic assessments, and ongoing staff training to maintain data integrity, confidentiality, and authenticity. Penetration testing should evolve from a periodic compliance task to a continuous, intelligent process embedded within development lifecycles and organizational culture to ensure resilient and trustworthy web applications.

7. Conclusion

This study examined existing research on penetration testing, with a specific focus on web application security. Furthermore, the study also analyzed current practices, challenges, and technological developments in web application penetration testing, with a particular focus on the role of automated tools. The findings indicate that while automated scanners provide significant advantages in terms of efficiency, scalability, and repeatability compared to manual methods, they are not equally effective across all vulnerability categories. Automated tools consistently identified common issues such as SQL injection, XSS, and configuration weaknesses, but they showed limitations in detecting logic-based vulnerabilities, multi-step attack chains, and client-side DOM-based attacks. These results reinforce the importance of adopting a hybrid testing strategy that combines multiple automated scanners with targeted manual analysis to achieve more comprehensive vulnerability coverage.

Additionally, the study highlights that different penetration testing tools vary widely in detection accuracy, feature sets, and reporting capabilities. As such, selecting an appropriate tool should be based on an organization's specific security needs, target environment, and the types of vulnerabilities most relevant to their risk profile. By mapping common vulnerabilities to suitable testing tools, this research supports more informed decision-making and contributes practical guidance to both practitioners and researchers in the field. Despite its contributions, this study has several limitations. First, it relies on existing literature and tool evaluations rather than conducting large-scale experimental testing across diverse real-world applications. Second, the effectiveness of penetration tools may vary depending on application architecture, technology stacks, and configuration contexts that were beyond the scope of this review. Finally, the rapidly evolving nature of web technologies means that new attack techniques and vulnerabilities may emerge that are not fully captured in current tools or in the studies reviewed, this work consolidates contemporary knowledge on web penetration testing, clarifies the strengths and weaknesses of automated tools, and offers a structured foundation for improving testing practices. Future research could expand on this by performing empirical tool benchmarking, integrating AI-driven detection methods, and evaluating hybrid manual-automated testing workflows across larger and more varied application environments. The aim of this paper is not only to propose a technical solution but also to highlight the key security challenges that modern web applications continue to face.

CRediT Authorship Contribution Statement

E. Bugingo: Conceptualization, Methodology, Writing – Original Draft. **V. Ishimwe:** Writing – Review & Editing, Formal analysis. **G. U. Simbi:** Writing – Review & Editing. **A. Dusenge:** Writing – Review & Editing. **J. B. Nizeyimana:** Writing – Review & Editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Declaration of Generative AI and AI-assisted Technologies in The Writing Process

The authors used generative AI to improve the writing clarity of this paper. They reviewed and edited the AI-assisted content and take full responsibility for the final publication.

References

[1] M. Mirjalili, A. Nowroozi, and M. Alidoosti, "A survey on a web penetration test," *Adv Comput Sci Int J*, vol. 3, pp. 117–121, 2014.

[2] Y. Sadqi and Y. Maleh, "A systematic review and taxonomy of web applications threats," *Inf. Secur. J. Glob. Perspect.*, vol. 31, pp. 1–27, 2022, doi: 10.1080/19393555.2021.1971731.
[3] E. Trickel and others, "Toss a Fault to Your Witcher: Applying Grey-box Coverage-Guided Mutational Fuzzing to Detect SQL and Command Injection Vulnerabilities," in *Proceedings of the 2023 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, 2023, pp. 2658–2675. doi: 10.1109/SP46215.2023.10179379.
[4] G. Deepa and P. S. Thilagam, "Securing web applications from injection and logic vulnerabilities: Approaches and challenges," *Inf. Softw. Technol.*, vol. 74, pp. 160–180, 2016, doi: 10.1016/j.infsof.2016.02.004.
[5] M. Alhamed and M. M. H. Rahman, "A Systematic Literature Review on Penetration Testing in Networks: Future Research Directions," *Appl. Sci.*, vol. 13, p. 6986, 2023, doi: 10.3390/app13126986.
[6] B. Mburano and W. Si, "Evaluation of Web Vulnerability Scanners Based on OWASP Benchmark," in *Proceedings of the 2018 26th International Conference on Systems Engineering (ICSEng)*, Sydney, Australia, 2018, pp. 1–6. doi: 10.1109/ICSENG.2018.8638197.
[7] Y. Makino and V. Klyuev, "Evaluation of web vulnerability scanners," in *Proceedings of the 2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, Warsaw, Poland, 2015, pp. 399–402. doi: 10.1109/IDAACS.2015.7340728.
[8] S. E. Idrissi, N. Berbiche, F. Guerouate, and M. Shibi, "Performance evaluation of web application security scanners for prevention and protection against vulnerabilities," *Int. J. Appl. Eng. Res.*, vol. 12, pp. 11068–11076, 2017.
[9] E. A. Altulaihan, A. Alismail, and M. Frikha, "A Survey on Web Application Penetration Testing," *Electronics*, vol. 12, no. 5, p. 1229, 2023, doi: 10.3390/electronics12051229.
[10] F. Kagorora, J. Li, D. Hanyurwimfura, and L. Camara, "Effectiveness of Web Application Security Scanners at Detecting Vulnerabilities behind AJAX/JSON," *Int. J. Innov. Res. Sci. Eng. Technol.*, vol. 4, pp. 4179–4188, 2015.
[11] H. J. Kam and J. J. Pauli, "Work in progress—web penetration testing: Effectiveness of student learning in Web application security," in *Proceedings of the 2011 Frontiers in Education Conference (FIE)*, Rapid City, SD, USA, Oct. 2011, pp. F3G–1.
[12] I. Mukhopadhyay, S. Goswami, and E. Mandal, "Web penetration testing using nessus and metasploit tool," *IOSR J Comput Eng*, vol. 16, pp. 126–129, 2014.
[13] M. Baykara, "Investigation and comparison of web application vulnerabilities test tools," *Int J Comput Sci Mob Comput IJCSMC*, vol. 7, pp. 197–212, 2018.
[14] R. M. Wibowo and A. Sulaksono, "Web vulnerability through cross site scripting (XSS) detection with OWASP security shepherd," *Indones J Inf Syst*, vol. 3, pp. 149–159, 2021, doi: 10.24002/ijis.v3i2.4192.
[15] F. Abu-Dabaseh and E. Alshammari, "Automated penetration testing: An overview," in *Proceedings of the 4th International Conference on Natural Language Computing*, Copenhagen, Denmark, Apr. 2018, pp. 121–129. doi: 10.5121/CSIT.2018.80610.
[16] L. K. Seng, N. Ithnin, and S. Z. M. Said, "The approaches to quantify web application security scanners quality: A review," *Int. J. Adv. Comput. Res.*, vol. 8, no. 38, pp. 285–312, 2018, doi: 10.19101/ijacr.2018.838012.
[17] E. Toch and others, "The privacy implications of cyber security systems," *ACM Comput. Surv.*, vol. 51, no. 2, pp. 1–27, 2018, doi: 10.1145/3172869.
[18] T. W. Thomas, M. Tabassum, B. Chu, and H. Lipford, "Security during application development," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018. doi: 10.1145/3173574.3173836.
[19] P. R. Vamsi and A. Jain, "Practical security testing of electronic commerce web applications," *Int. J. Adv. Netw. Appl.*, vol. 13, no. 1, pp. 4861–4873, 2021, doi: 10.35444/IJANA.2021.13109.
[20] P. R. Vamsi and A. Jain, "Getting started with android mobile applications security testing." [Online]. Available: https://journal.scsa.ge/papers/getting-started-with-android-mobile-applications-security-testing/
[21] R. Kumar, S. Sharma, and V. Patel, "AI-Driven Solutions for Phishing Detection: A Comparative Analysis," *Cybersecurity Priv.*, vol. 12, no. 4, pp. 78–95, 2024.
[22] A. K. Priyanka and S. S. Smruthi, "Web application vulnerabilities: Exploitation and prevention," in *2020 International Conference on Electrotechnical Complexes and Systems (ICOECS)*, 2020, pp. 729–734. doi: 10.1109/icoecs50468.2020.9278437.
[23] K. Amin and P. Sharma, "Red team analysis of information security measures and response." [Online]. Available: https://www.academia.edu/download/64372201/IRJET-V7I4823.pdf
[24] P. Vats, M. Mandot, and A. Gosain, "A comprehensive literature review of penetration testing & its applications," in *2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, 2020, pp. 674–680. doi: 10.1109/icrito48877.2020.9197961.
[25] A. Goutam and V. Tiwari, "Vulnerability assessment and penetration testing to enhance the security of web application," in *2019 4th International Conference on Information Systems and Computer Networks (ISCON)*, 2019, pp. 601–605. doi: 10.1109/iscon47742.2019.9036175.
[26] J. N. Goel and B. M. Mehtre, "Vulnerability assessment & penetration testing as a cyber defence technology," *Procedia Comput. Sci.*, vol. 57, pp. 710–715, 2015, doi: 10.1016/j.procs.2015.07.458.
[27] Y. Khera, D. Kumar, Sujay, and N. Garg, "Analysis and impact of vulnerability assessment and penetration testing," in *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, 2019. doi: 10.1109/comitcon.2019.8862224.
[28] A. Hasan and D. Meva, "Web application safety by penetration testing." [Online]. Available: https://www.academia.edu/download/58290599/SSRN-id3315587.pdf
[29] I. Yaqoob, S. A. Hussain, S. Mamoon, N. Naseer, J. Akram, and A. UR Rehman, "Penetration testing and vulnerability assessment," *J. Netw. Commun. Emerg. Technol. JNCET*, vol. 7, no. 8, pp. 10–18, 2017.
[30] A. M. Hasan, D. T. Meva, A. K. Roy, and J. Doshi, "Perusal of web application security approach," in *2017 International Conference on Intelligent Communication and Computational Techniques (ICCT)*, 2017, pp. 90–95. doi: 10.1109/intelcct.2017.8324026.
[31] M. Z. Hussain, M. Z. Hasan, M. Taimoor, and A. Chughtai, "Penetration testing in system administration," *Int. J. Sci. Technol. Res.*, vol. 6, no. 6, pp. 275–278, 2017.
[32] M. F. Haque, M. B. A. Miah, and F. A. Masud, "Enhancement of web security against external attack," *Eur. Sci. J.*, vol. 13, no. 15, p. 228, 2017, doi: 10.19044/esj.2017.v13n15p228.
[33] S. Nagpure and S. Kurkure, "Vulnerability assessment and penetration testing of web application," in *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)*, 2017. doi: 10.1109/iccubea.2017.846392020.

[34] P. S. Shinde and S. B. Ardhapurkar, "Cyber security analysis using vulnerability assessment and penetration testing," in *2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare (Startup Conclave)*, 2016. doi: 10.1109/startup.2016.7583912.

[35] D. Nyambo, Z. Yonah, and C. Tarimo, "On the identification of required security controls suitable for converged web and mobile applications," *Int. J. Comput. Digit. Syst.*, vol. 5, no. 1, 2016, doi: 10.12785/ijcds/050105.

[36] H. Singh, J. Surender, and K. V. Pankaj, "Penetration testing: Analyzing the security of the network by Hacker's mind," *Vol. V IJLTEMAS*, pp. 56–60, 2016.

[37] A. Lamba, "Cyber Attack prevention using VAPT tools (vulnerability assessment & penetration testing)." [Online]. Available: http://www.cikitusi.com/gallery/9-996.pdf

[38] S. Shah and B. M. Mehtre, "An overview of vulnerability assessment and penetration testing techniques," *J. Comput. Virol. Hacking Tech.*, vol. 11, no. 1, pp. 27–49, 2014, doi: 10.1007/s11416-014-0231-x.

[39] S. Shah and B. M. Mehtre, "A reliable strategy for proactive self-defence in cyber space using VAPT tools and techniques," in *2013 IEEE International Conference on Computational Intelligence and Computing Research*, 2013. doi: 10.1109/iccic.2013.6724216.

[40] H. S. Lallie and others, "Cyber security in the age of COVID-19: A timeline and analysis of cyber-crime and cyber-attacks during the pandemic," *Comput. Secur.*, vol. 105, p. 102248, 2021, doi: 10.1016/j.cose.2021.102248.

[41] B. Kumar and S. Roy, "An empirical study on usability and security of E-commerce websites," *Advances in Intelligent Systems and Computing*. pp. 735–746, 2021. doi: 10.1007/978-981-15-7527-3_69.

[42] M. Humayun, M. Niazi, N. Jhanjhi, M. Alshayeb, and S. Mahmood, "Cyber security threats and vulnerabilities: A systematic mapping study," *Arab. J. Sci. Eng.*, vol. 45, no. 4, pp. 3171–3189, 2020, doi: 10.1007/s13369-019-04319-2.

[43] O. B. Fredj, O. Cheikhrouhou, M. Krichen, H. Hamam, and A. Derhab, "An OWASP top ten driven survey on web application protection methods," *Risks and Security of Internet and Systems*. Springer, Cham, Switzerland, pp. 235–252, 2021. doi: 10.1007/978-3-030-68887-5_14.

[44] K. Pareek, "A study of web application penetration testing," *IJITE*, vol. 7, pp. 1776–2321, 2019.

[45] N. Auricchio, A. Cappuccio, F. Caturano, G. Perrone, and S. P. Romano, "An automated approach to web offensive security," *Comput Commun*, vol. 195, pp. 248–261, 2022, doi: 10.1016/j.comcom.2022.08.018.

[46] A. Alanda, D. Satria, M. Ardhana, A. A. Dahlan, and H. A. Mooduto, "Web application penetration testing using SQL Injection attack," *JOIV Int J Inf. Vis*, vol. 5, pp. 320–326, 2021, doi: 10.30630/joiv.5.3.470.

[47] J. K. Alhassan, S. Misra, A. Umar, R. Maskeliūnas, R. Damaševičius, and A. Adewumi, "A fuzzy classifier-based penetration testing for web applications," *Advances in Intelligent Systems and Computing*. Springer, Cham, Switzerland, pp. 95–104, 2018. doi: 10.1007/978-3-319-73450-7_10.

[48] M. Albahar, D. Alansari, and A. Jurcut, "An empirical comparison of pen-testing tools for detecting web app vulnerabilities," *Electronics*, vol. 11, p. 2991, 2022.

[49] D. Hyka and A. Basholli, "Some Maple algorithms generating diagnostically strong cryptographic examples," *Int. J. Cryptogr. Inf. Secur.*, vol. 4, no. 1, pp. 11–18, 2014.

[50] D. Hyka and A. Basholli, "Some Cryptographic Weakness and the Ways to Avoid Them in ICT Services of Developing Countries," *Int. J. Inf. Secur. Sci.*, vol. 3, no. 2, pp. 173–182, 2014.

[51] OWASP (Open Web Application Security Project), "OWASP Top 10."

[52] A. Hyra, F. Basholli, M. B, and A. Basholli, "Data Security in Public and Private Administration: Challenges, Trends, and Effective Protection in the Era of Digitalization," *Adv. Eng. Days*, vol. 7, pp. 125–127, 2023.

[53] Framework Security, "Automated vs Manual vs Hybrid Penetration Tests: Which Approach is Right for Your Organization?." [Online]. Available: https://frameworksec.com/post/automated-vs-manual-vs-hybrid-pen-tests-which-approach-is-right-for-your-organization

[54] S. Qadir, E. Waheed, A. Khanum, and S. Jehan, "Comparative Evaluation of Approaches & Tools for Effective Security Testing of Web Applications," 2025.

[55] K. Abdulghaffar, N. Elmrabit, and M. Yousefi, "Enhancing Web Application Security through Automated Penetration Testing with Multiple Vulnerability Scanners," *Computers*, vol. 12, no. 11, p. 235, 2023, doi: 10.3390/computers12110235.

[56] W. Wardana, A. Almaarif, and A. Widjajarto, "Vulnerability assessment and penetration testing on the xyz website using NIST 800-115 standard," *J Ilm Indones*, vol. 7, pp. 520–529, 2022, doi: 10.36418/SYNTAX-LITERATE.V7I1.5800.

[57] A. Sołtysik-Piorunkiewicz and M. Krysiak, "The cyber threats analysis for web applications security in industry 4.0," *Recent Advances in Computational Optimization*. Springer Science, Business Media LLC, Cham, Switzerland, pp. 127–141, 2020. doi: 10.1007/978-3-030-40417-8_8.

[58] M. Alenezi, A. Agrawal, R. Kumar, and R. A. Khan, "Evaluating Performance of web application security through a fuzzy based hybrid multi-criteria decision-making approach: Design tactics perspective," *IEEE Access*, vol. 8, pp. 25543–25556, 2020, doi: 10.1109/ACCESS.2020.2971444.

[59] L. Qi and others, "An exception handling approach for privacy-preserving service recommendation failure in a cloud environment," *Sensors*, vol. 18, no. 7, p. 2037, 2018, doi: 10.3390/s18072037.

[60] R. Gupta, "An innovative security strategy using reactive web application honeypot," *ArXiv Prepr.*, 2020.

[61] D. Priyawati, S. Rokhmah, and I. C. Utomo, "Website vulnerability testing and analysis of website application using OWASP," *Int. J. Comput. Inf. Syst. IJCIS*, vol. 3, no. 3, pp. 142–147, 2022, doi: 10.29099/ijcis.v3i3.77.

[62] M. Willberg, "Web Application Security Testing with Owasp Top 10 Frameworks," Bachelor's Thesis, Turku, Finland, 2019.

[63] T. Lauinger, A. Chaabane, S. Arshad, W. Robertson, C. Wilson, and E. Kirda, "Thou shalt not depend on me: Analysing the use of outdated javascript libraries on the web," *ArXiv Prepr.*, 2017.

[64] H. Shahriar and M. Zulkernine, "Information-theoretic detection of SQL injection attacks," in *Proceedings of the 2012 IEEE 14th International Symposium on High-Assurance Systems Engineering*, Omaha, NE, USA, Oct. 2012, pp. 40–47. doi: 10.1109/HASE.2012.12.

[65] N. Elisa, "Usability, accessibility and web security assessment of e-government websites in Tanzania," *Int. J. Comput. Appl.*, vol. 164, pp. 42–48, 2017, doi: 10.5120/ijca2017913648.

[66] A. Tundis, W. Mazurczyk, and M. Mühlhäuser, "A Review of Network Vulnerabilities Scanning Tools: Types, Capabilities, and Functions," in *Proceedings of the 13th International Conference on Availability, Reliability and Security*, Hamburg, Germany, 2018, pp. 1–10. doi: 10.1145/3230833.3233287.

[67] S. Bennetts, "OWASP Zed Attack Proxy." San Francisco, CA, USA, 2013.

[68] M. Alsaleh, N. Alomar, M. Alshreef, A. Alarifi, and A. Al-Salman, "Performance-Based Comparative Assessment of Open Source Web Vulnerability Scanners," *Secur. Commun. Netw.*, p. 6158107, 2017, doi: 10.1155/2017/6158107.

[69] R. Amankwah, J. Chen, P. K. Kudjo, and D. Towey, "An Empirical Comparison of Commercial and Open-Source Web Vulnerability Scanners," *Softw. Pract. Exp.*, vol. 50, pp. 1842–1857, 2020, doi: 10.1002/spe.2810.

[70] M. Curphey and others, “A Guide to Building Secure Web Applications,” technical report, Sept. 2002.
[71] D. Scott and R. Sharp, “Abstracting Application-Level Web Security,” in *The 11th International Conference on the World Wide Web*, Honolulu, Hawaii, May 2002, pp. 396–407. doi: 10.1145/511446.511498.
[72] D. Scott and R. Sharp, “Developing Secure Web Applications,” *IEEE Internet Comput.*, vol. 6, no. 6, pp. 38–45, Nov. 2002, doi: 10.1109/MIC.2002.1067735.
[73] Sanctum Inc., “AppShield 4.0 Whitepaper.” [Online]. Available: http://www.sanctuminc.com/
[74] Kavado, Inc., “InterDo Version 3.0.” 2003.
[75] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood, “Deep Packet Inspection Using Parallel Bloom Filters,” in *Proceedings of the 11th Symposium on High Performance Interconnects*, Stanford, California, 2003, pp. 44–51.
[76] M. Bobbitt, “Bulletproof Web Security.” [Online]. Available: http://infosecuritymag.techtarget.com/2002/may/bulletproof.shtml
[77] R. S. Sandhu, “Lattice-Based Access Control Models,” *IEEE Comput.*, vol. 26, no. 11, pp. 9–19, 1993, doi: 10.1109/2.241422.
[78] J. A. Goguen and J. Meseguer, “Security Policies and Security Models,” in *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, California, Apr. 1982, pp. 11–20. doi: 10.1109/SP.1982.10014.
[79] J. Joshi, W. Aref, A. Ghafoor, and E. Spafford, “Security Models for Web-Based Applications,” *Commun. ACM*, vol. 44, no. 2, pp. 38–44, Feb. 2001, doi: 10.1145/359205.359224.
[80] Sanctum Inc., “Web Application Security Testing – AppScan 3.5.” [Online]. Available: http://www.sanctuminc.com/
[81] Mozilla.org, “Mozilla Layout Engine.” [Online]. Available: http://www.mozilla.org/newlayout/
[82] R. C. Miller and K. Bharat, “SPHINX: A Framework for Creating Personal, Site-Specific Web Crawlers,” in *Proceedings of the 7th International World Wide Web Conference*, Brisbane, Australia, Apr. 1998, pp. 119–130. doi: 10.1016/S0169-7552(98)00064-6.
[83] U. Manber, M. Smith, and B. Gopal, “WebGlimpse—Combining Browsing and Searching,” in *Proceedings of the USENIX 1997 Annual Technical Conference*, Anaheim, California, Jan. 1997.
[84] J. Cho and H. Garcia-Molina, “Parallel Crawlers,” in *Proceedings of the 11th International Conference on the World Wide Web*, Honolulu, Hawaii, May 2002, pp. 124–135. doi: 10.1145/511446.511464.
[85] N. Antunes and M. Vieira, “Penetration testing for web services,” *Computer*, vol. 47, pp. 30–36, 2013, doi: 10.1109/MC.2013.48.
[86] S. Alazmi and D. C. De Leon, “A Systematic Literature Review on the Characteristics and Effectiveness of Web Application Vulnerability Scanners,” *IEEE Access*, vol. 10, pp. 33200–33219, 2022, doi: 10.1109/ACCESS.2022.3161746.
[87] N. Albalawi, N. Alamrani, R. Aloufi, M. Albalawi, A. Aljaedi, and A. R. Alharbi, “The Reality of Internet Infrastructure and Services Defacement: A Second Look at Characterizing Web-Based Vulnerabilities,” *Electronics*, vol. 12, p. 2664, 2023, doi: 10.3390/electronics12122664.
[88] J. Shahid, M. K. Hameed, I. T. Javed, K. N. Qureshi, M. Ali, and N. Crespi, “A Comparative Study of Web Application Security Parameters: Current Trends and Future Directions,” *Appl. Sci.*, vol. 12, p. 4077, 2022, doi: 10.3390/app12094077.
[89] T. Laskos, “Arachni—Web Application Security Scanner Framework.” [Online]. Available: https://github.com/Arachni
[90] “ZAPping the OWASP Top 10.” [Online]. Available: https://www.zaproxy.org/docs/guides/zapping-the-top-10-2021/
[91] J. Hance, J. Milbrath, N. Ross, and J. Straub, “Distributed Attack Deployment Capability for Modern Automated Penetration Testing,” *Computers*, vol. 11, p. 33, 2022, doi: 10.3390/computers11030033.
[92] X. Qiu, S. Wang, Q. Jia, C. Xia, and Q. Xia, “An automated method of penetration testing,” in *Proceedings of the 2014 IEEE Computers, Communications and IT Applications Conference*, Beijing, China, 2014, pp. 211–216. doi: 10.1109/ComComIT.2014.7017164.
[93] T.-Y. Zhou, Y.-C. Zang, J.-H. Zhu, and Q.-X. Wang, “NIG-AP: A new method for automated penetration testing,” *Front. Inf. Technol. Electron. Eng.*, vol. 20, pp. 1277–1288, 2019, doi: 10.1631/FITEE.1800752.
[94] T. H. M. Le, D. Hin, R. Croft, and M. Ali Babar, “DeepCVA: Automated Commit-level Vulnerability Assessment with Deep Multi-task Learning,” in *Proceedings of the 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2021, pp. 717–729. doi: 10.1109/ASE51524.2021.9678505.
[95] Y. W. Huang, S. K. Huang, T. P. Lin, and C. H. Tsai, “Web Application Security Assessment by Fault Injection and Behavior Monitoring,” in *Proceedings of the Twelfth International World Wide Web Conference*, Budapest, Hungary, May 2003, pp. 148–159. doi: 10.1145/775152.775174.
[96] Y. W. Huang, C. H. Tsai, D. T. Lee, and S. Y. Kuo, “Non-Detrimental Web Application Security Auditing,” in *Proceedings of the Fifteenth IEEE International Symposium on Software Reliability Engineering (ISSRE2004)*, Rennes and Saint-Malo, France, Nov. 2004. doi: 10.1109/ISSRE.2004.25.